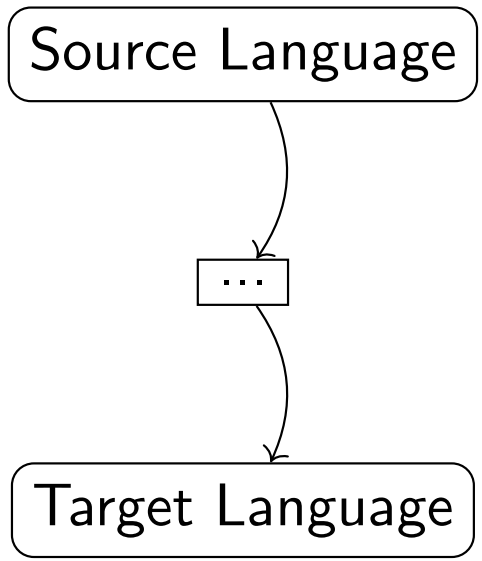
Enforcing Well-bracketed Control Flow and Stack Encapsulation using Linear Capabilities

Lau Skorstengaard[1] Dominique Devriese[2] Lars Birkedal[1]

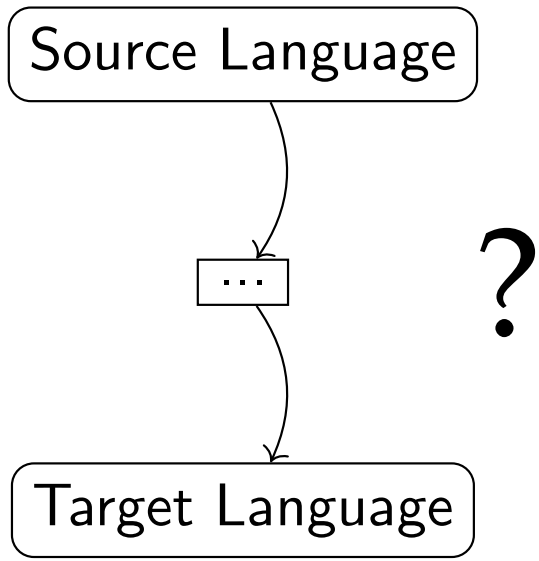
[1]Aarhus University [2]KU Leuven

PriSC 2018, Los Angeles

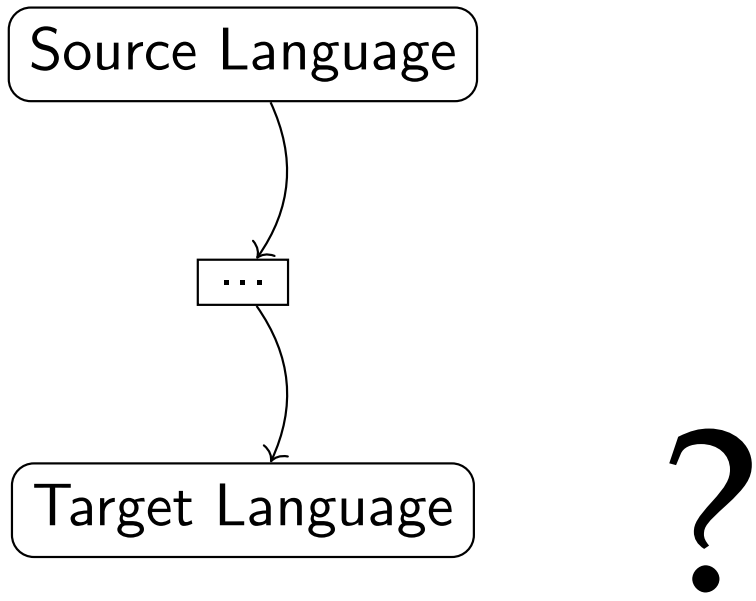
A realistic secure compiler?



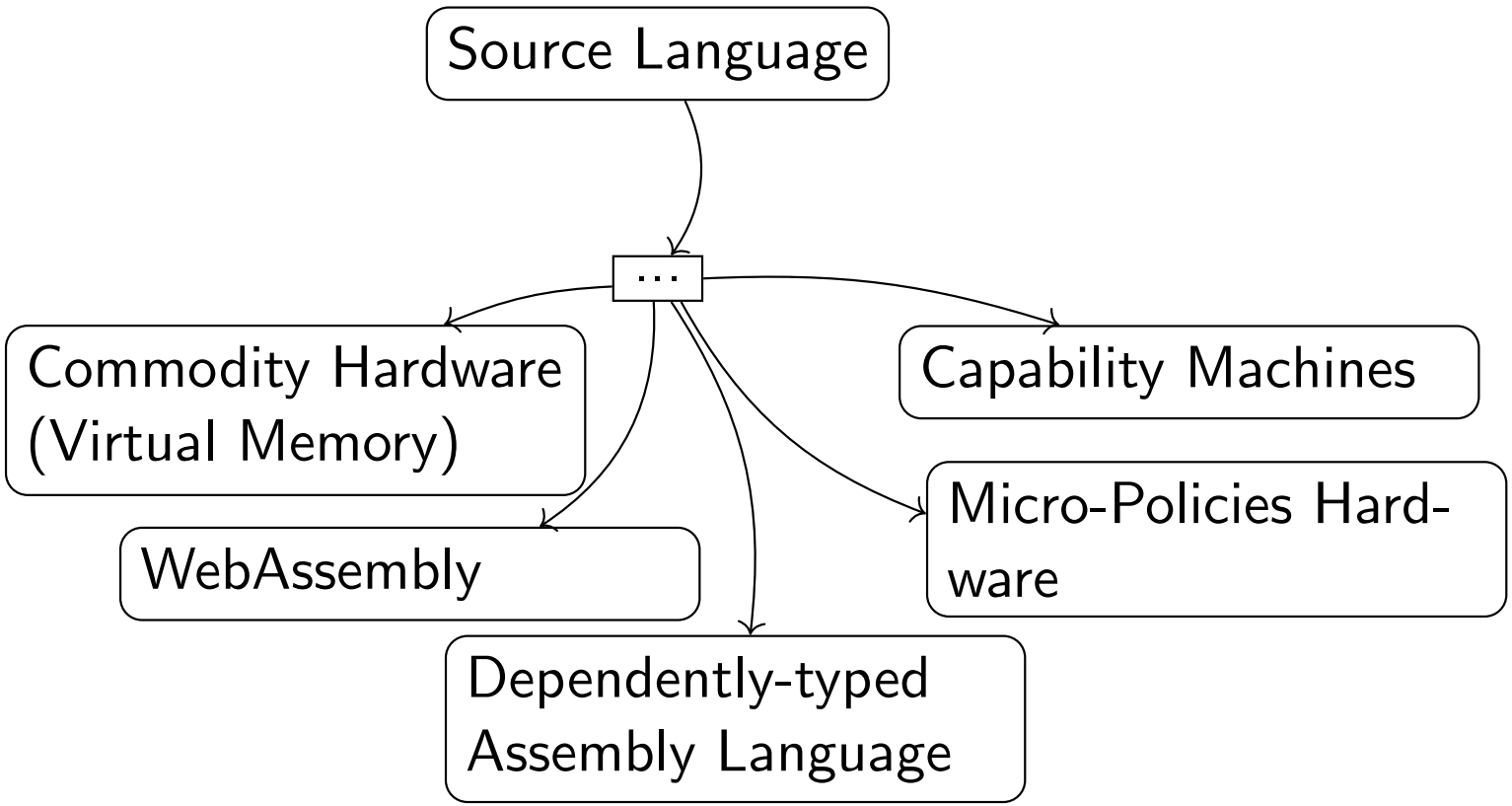
A realistic secure compiler?



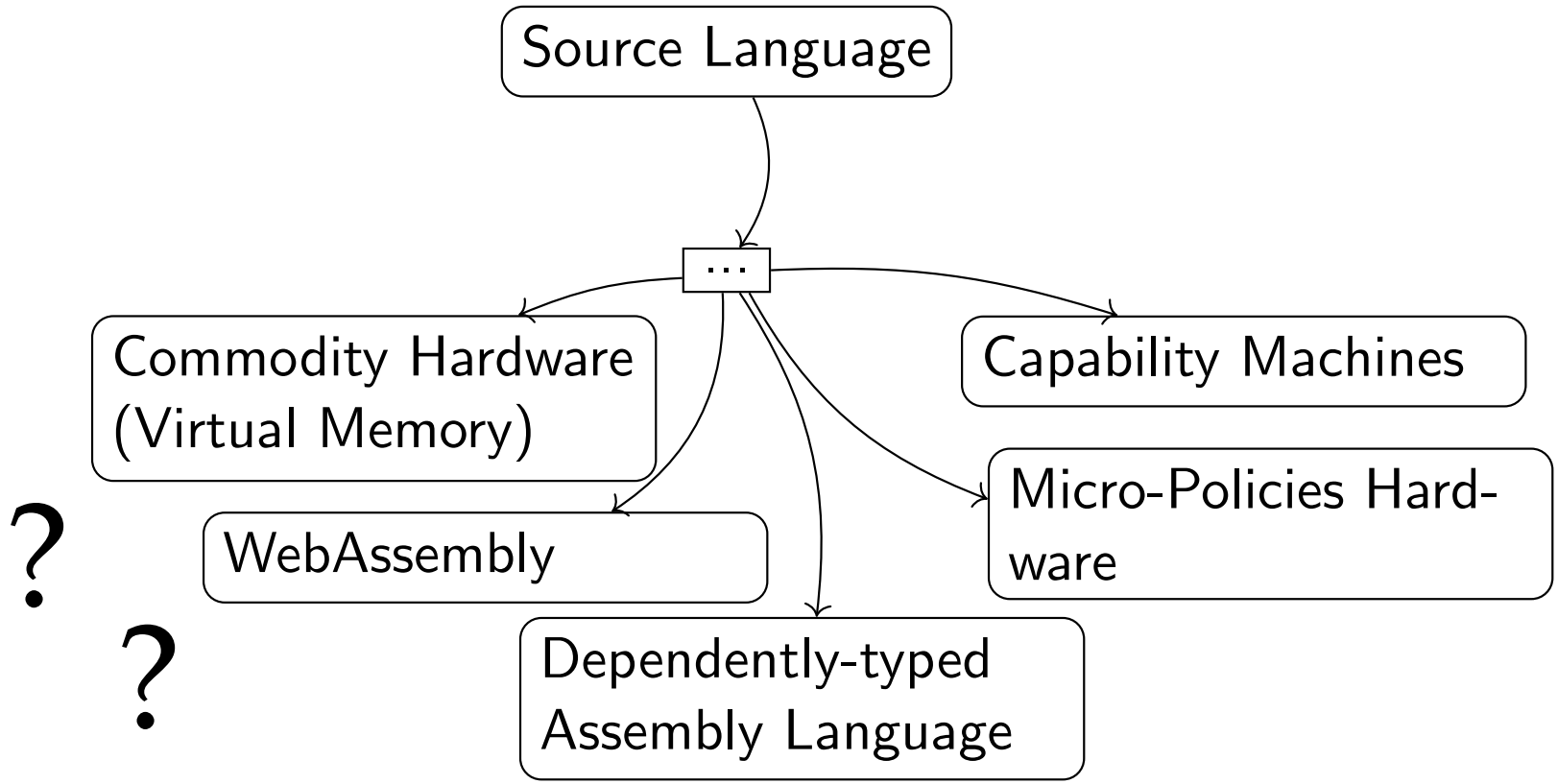
A realistic secure compiler?



A realistic secure compiler?

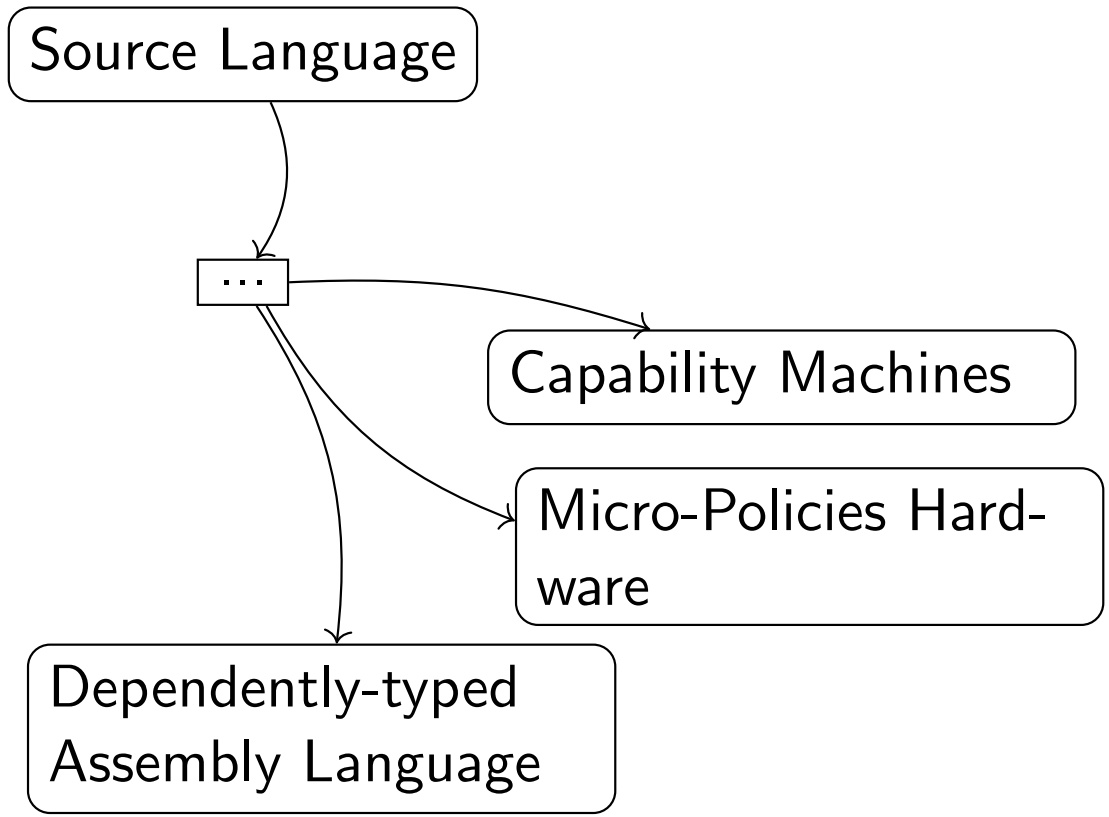


A realistic secure compiler?

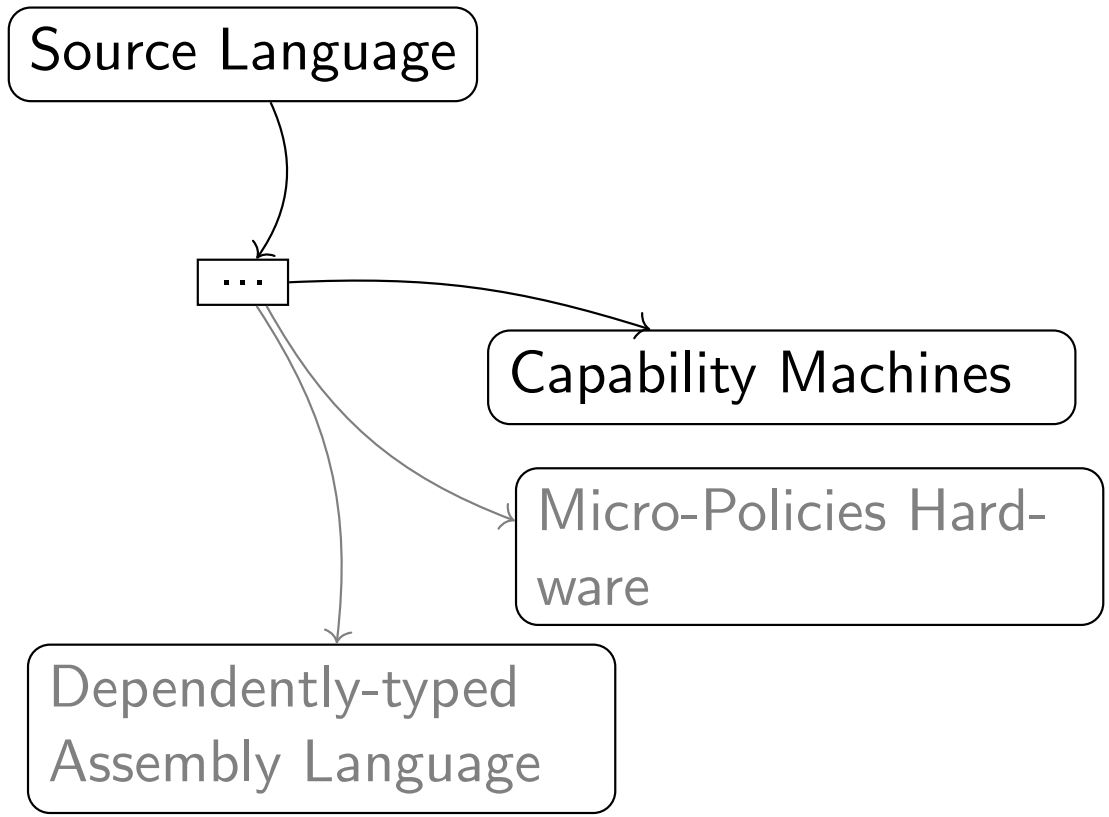


Fine-grained encapsulation?

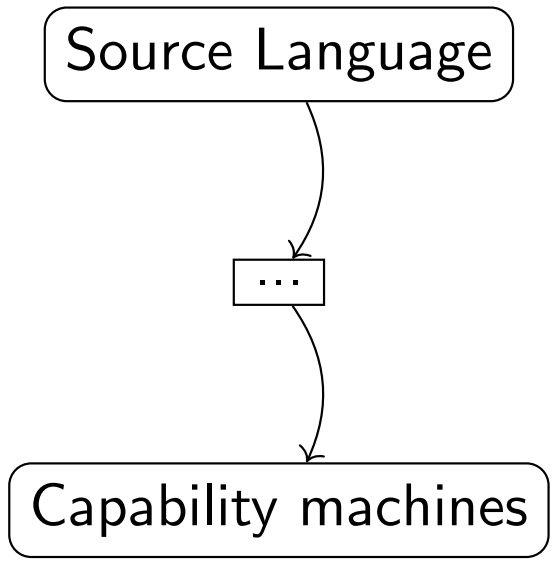
A realistic secure compiler?



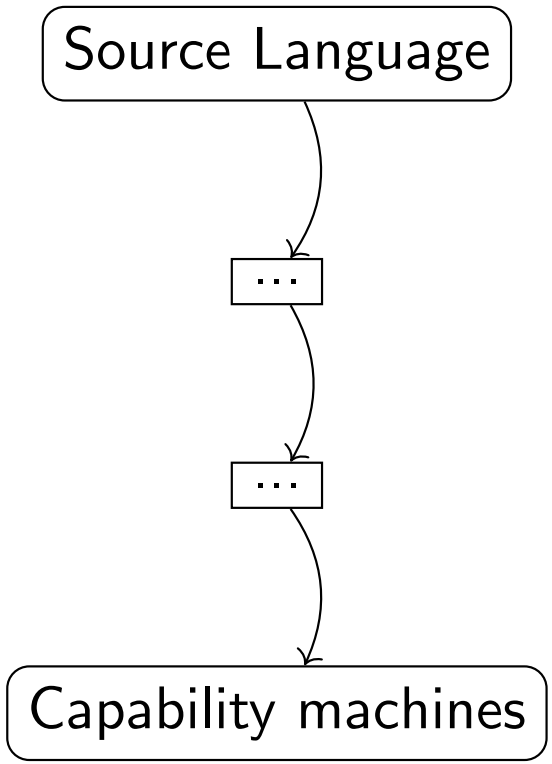
A realistic secure compiler?



A realistic secure compiler?

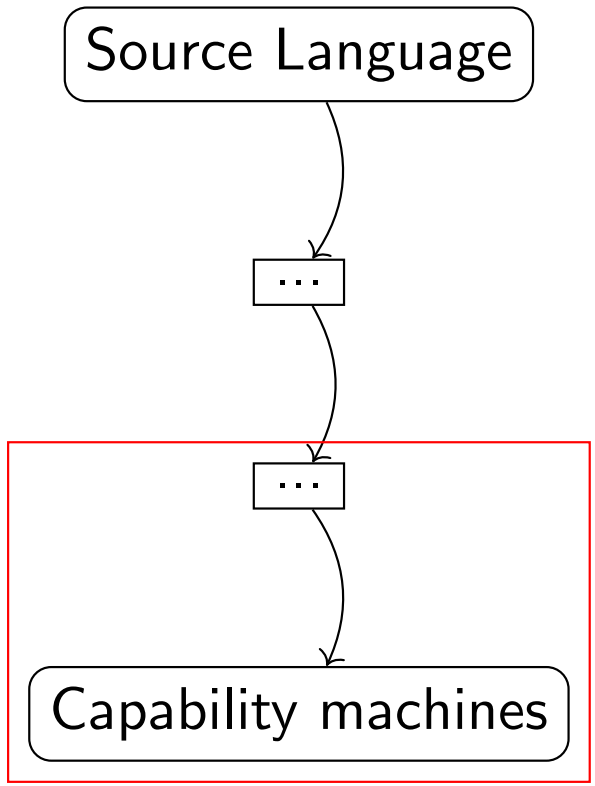


A realistic secure compiler?



Well-bracketed control flow, stack encapsulation?

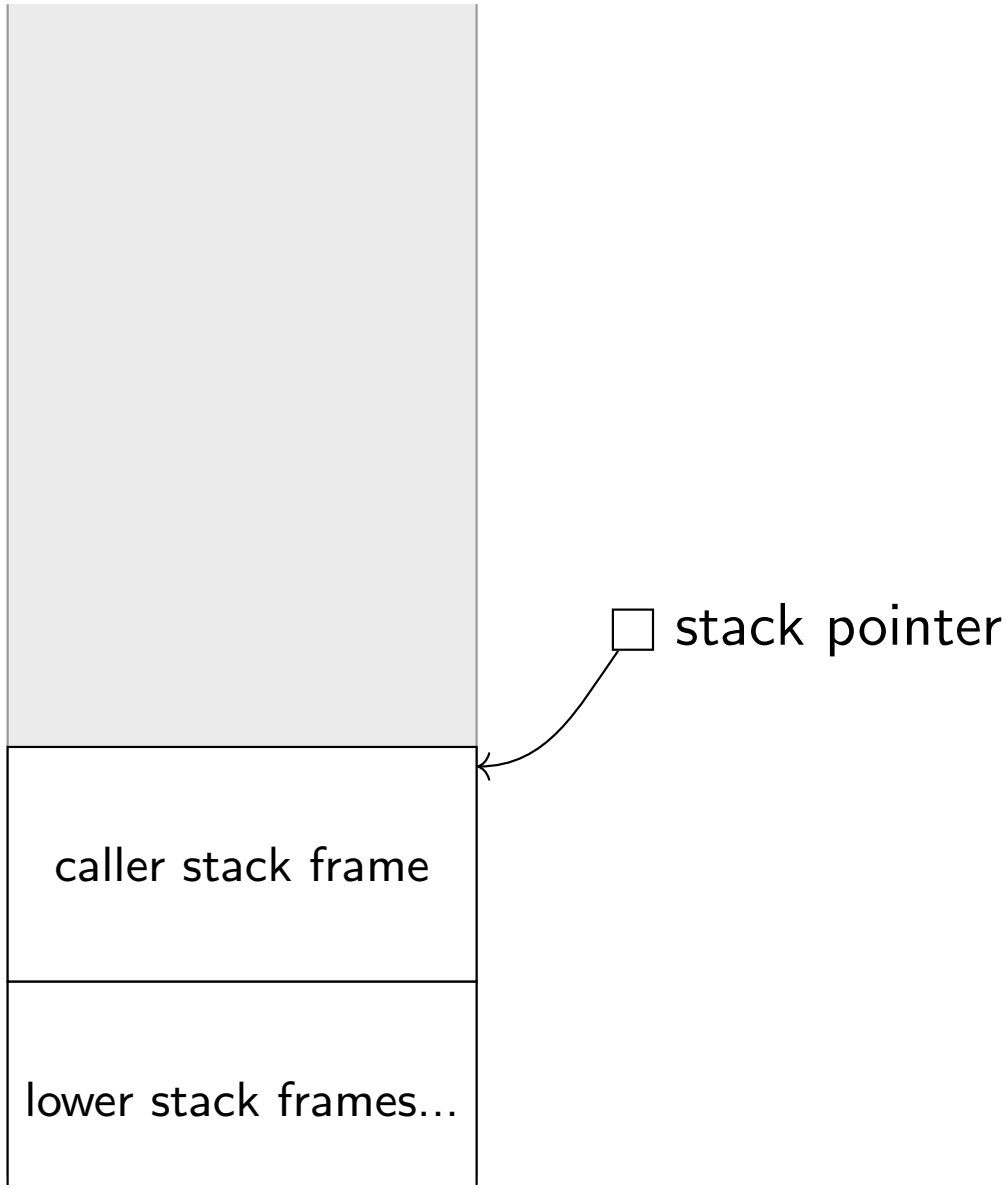
A realistic secure compiler?



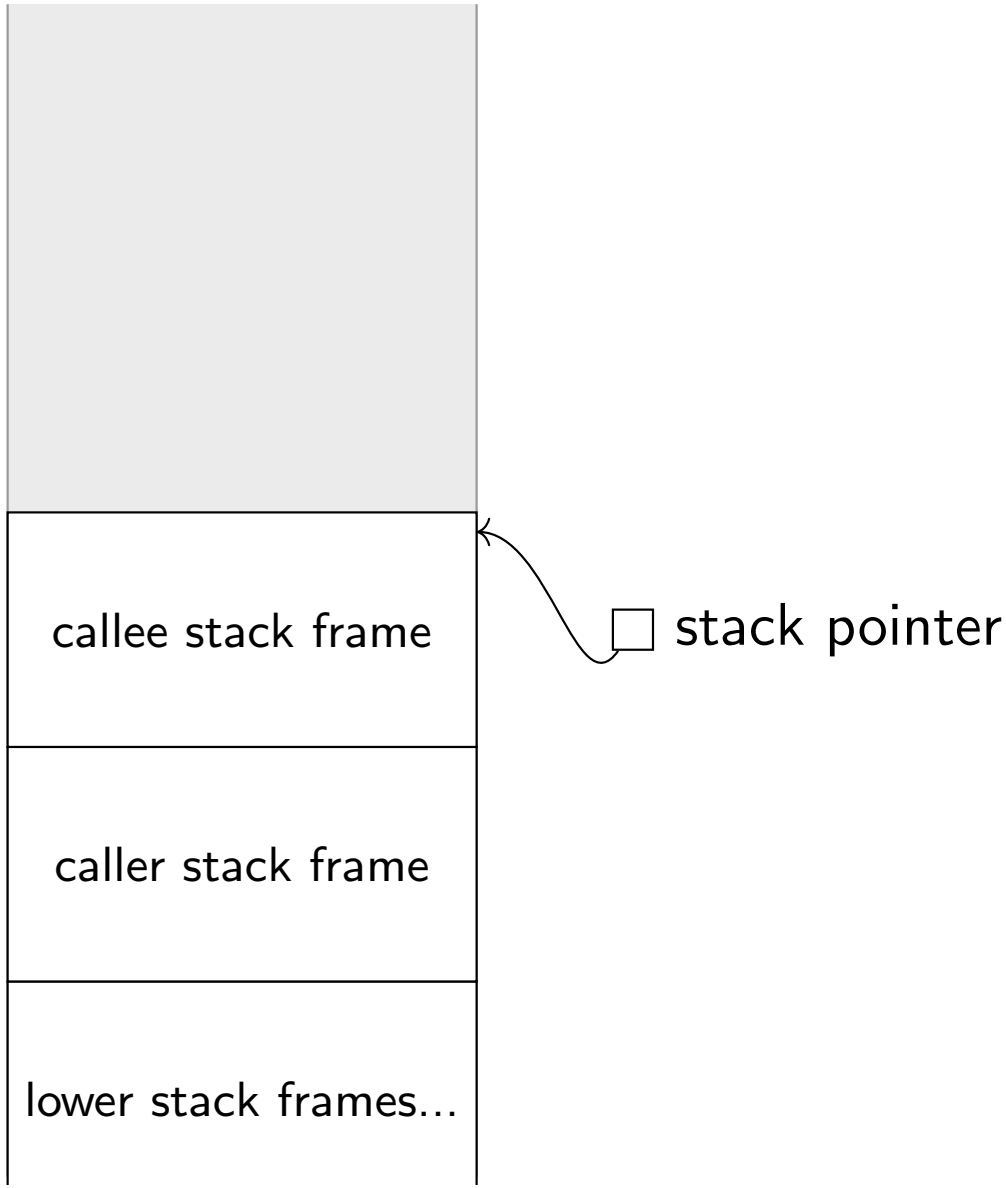
Let's take a look!

Outline

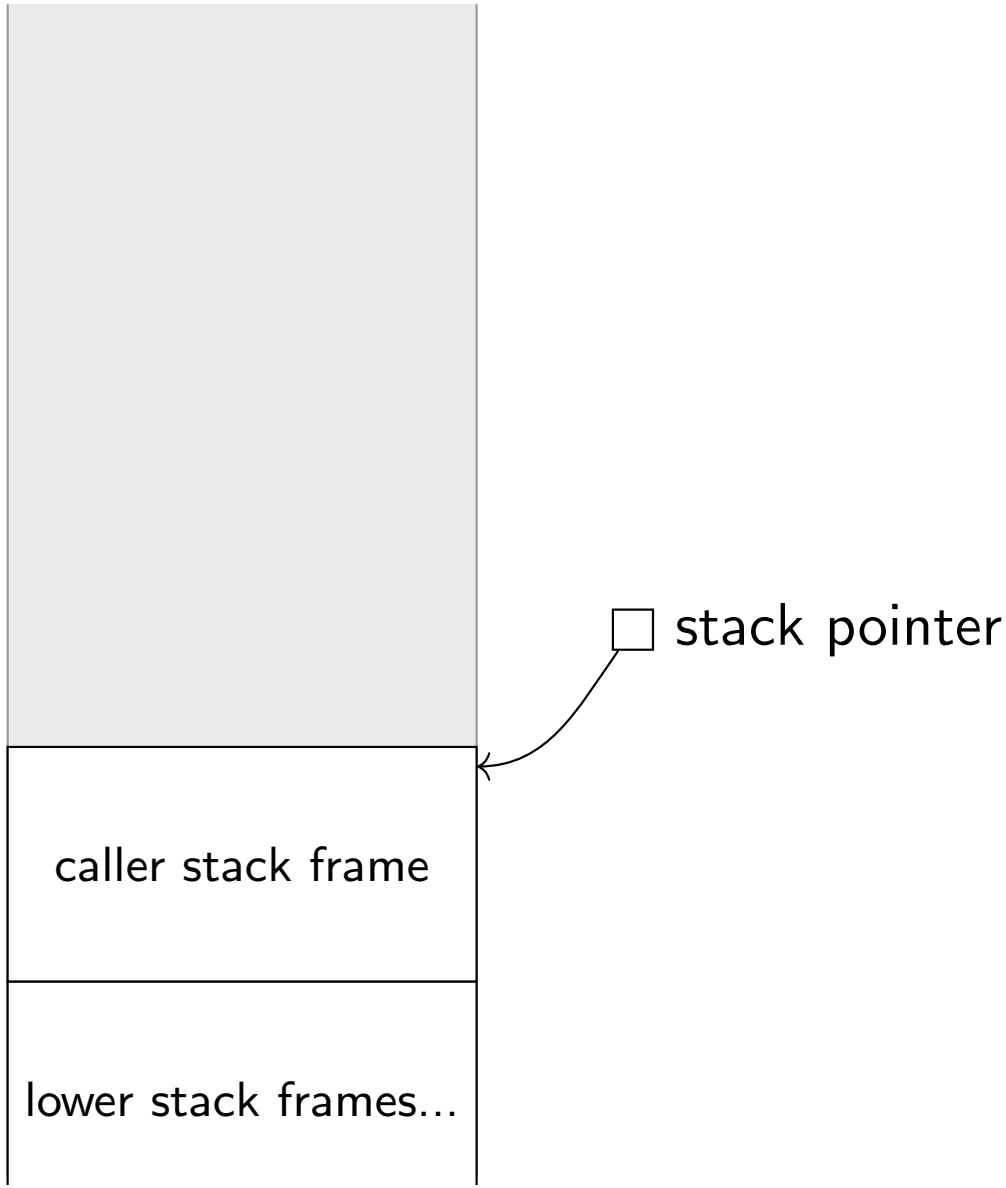
Traditional stack pointers



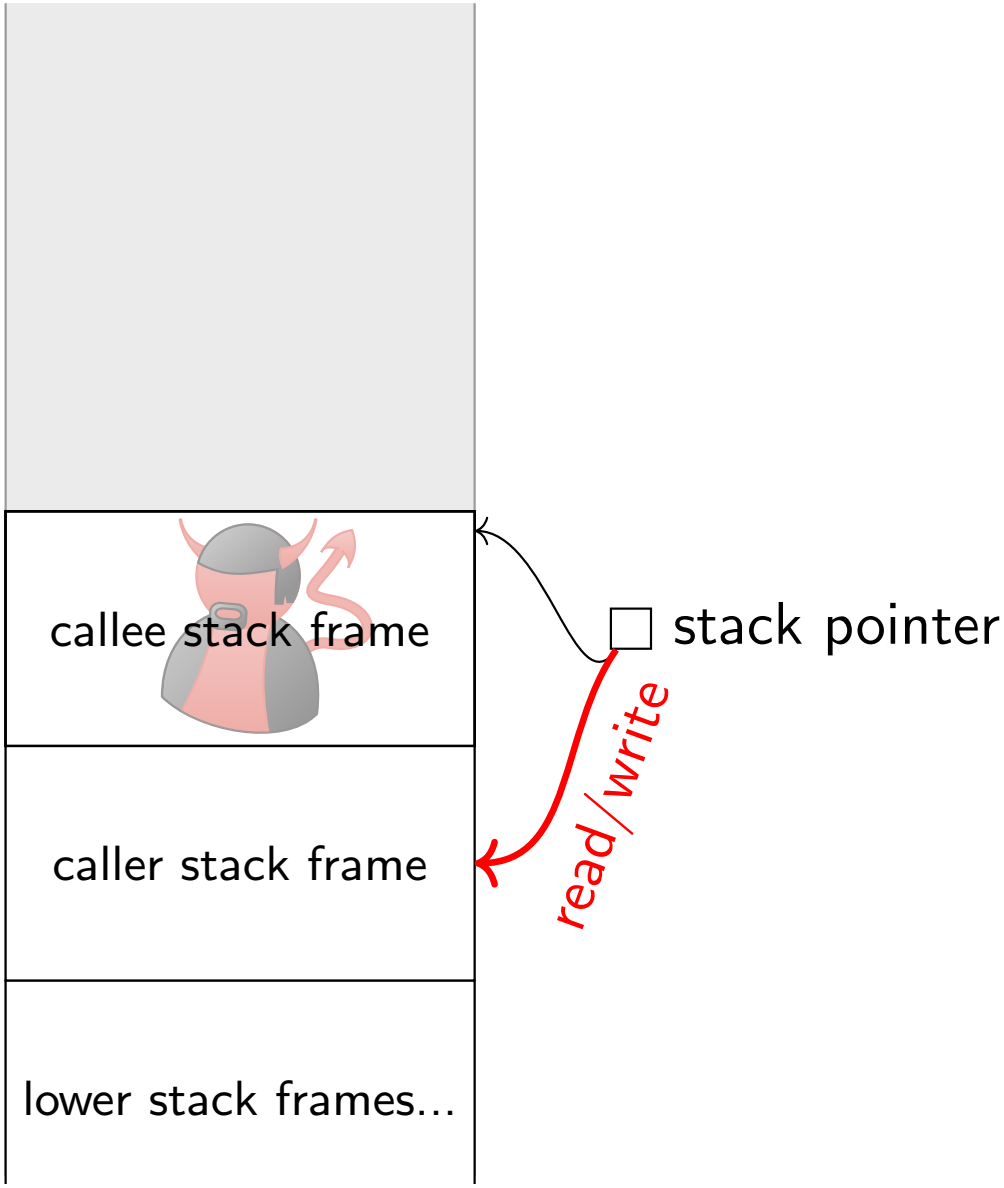
Traditional stack pointers



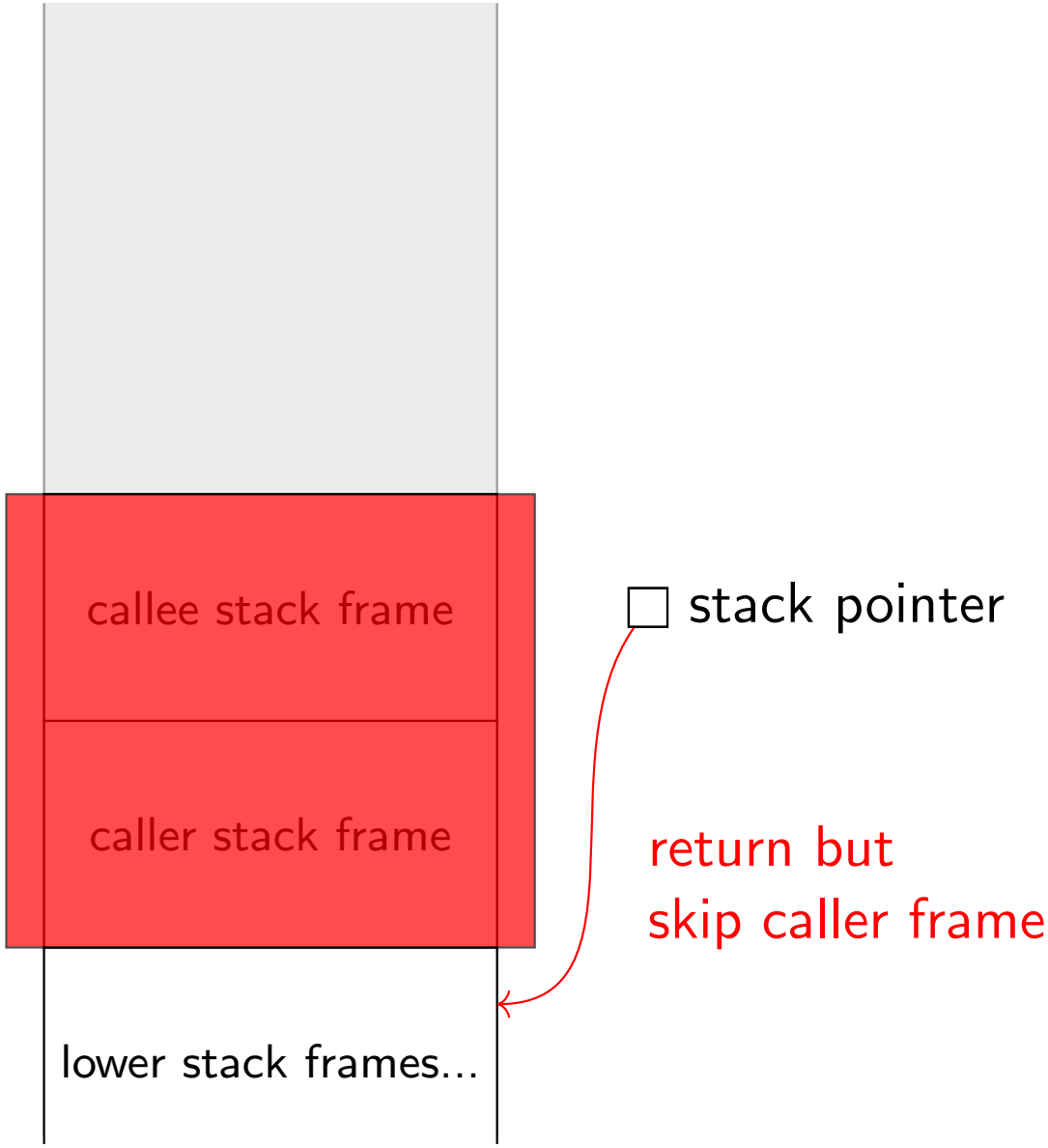
Traditional stack pointers



Traditional stack pointers



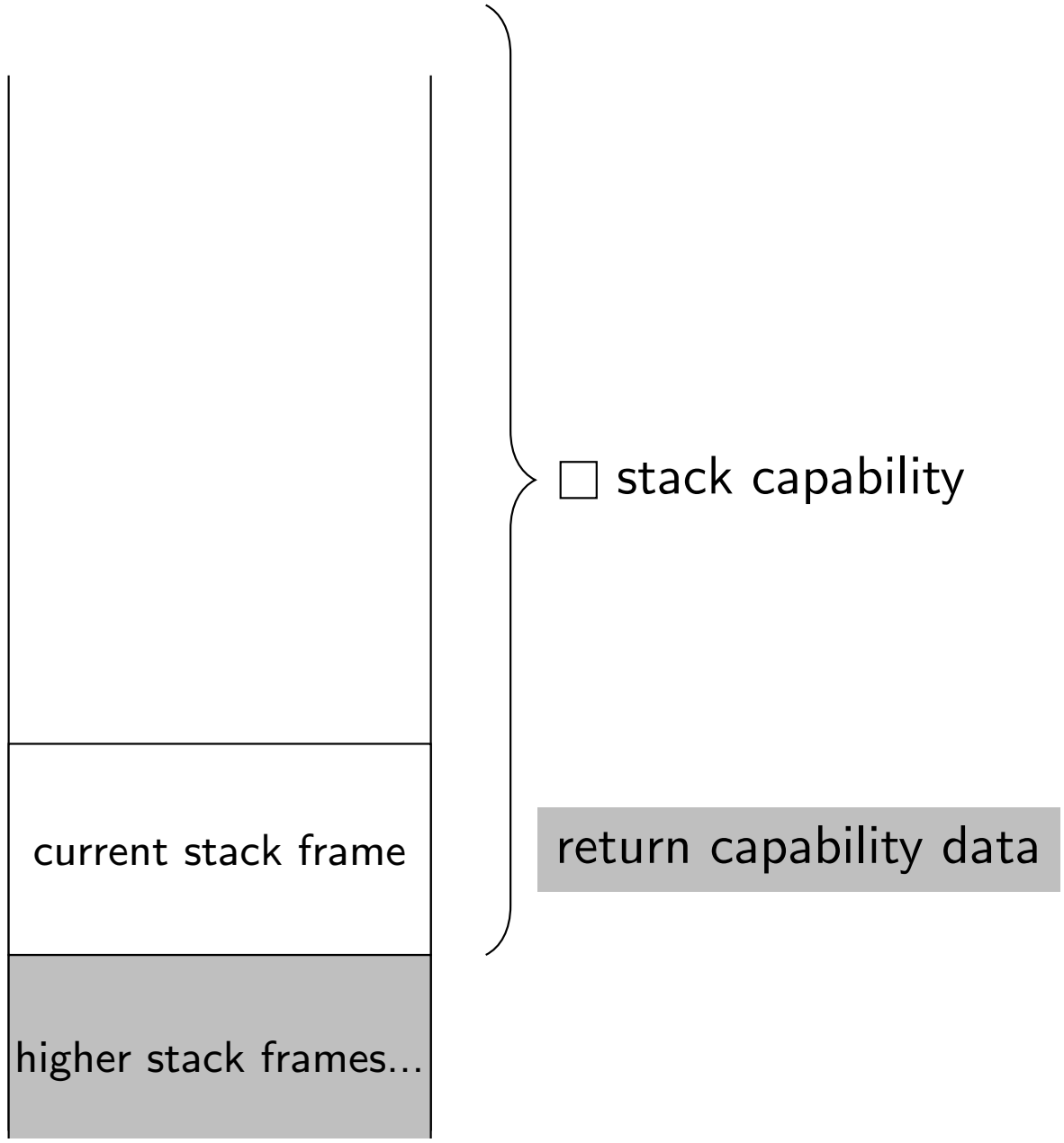
Traditional stack pointers



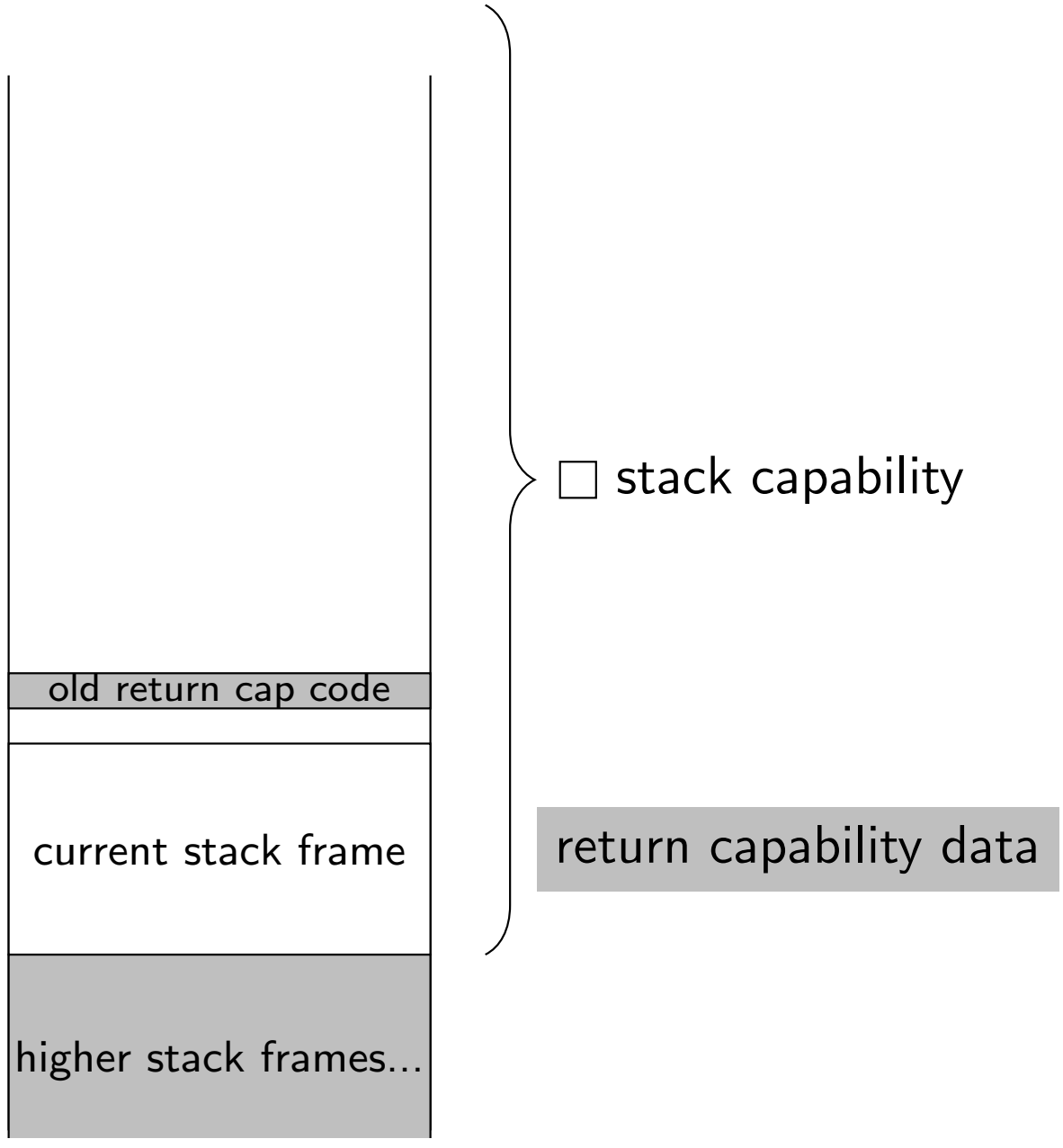
Stack and return pointer security

- Desirable properties:
  - Well-bracketed control flow
  - Stack frame encapsulation
- How to enforce?
  - Capabilities?

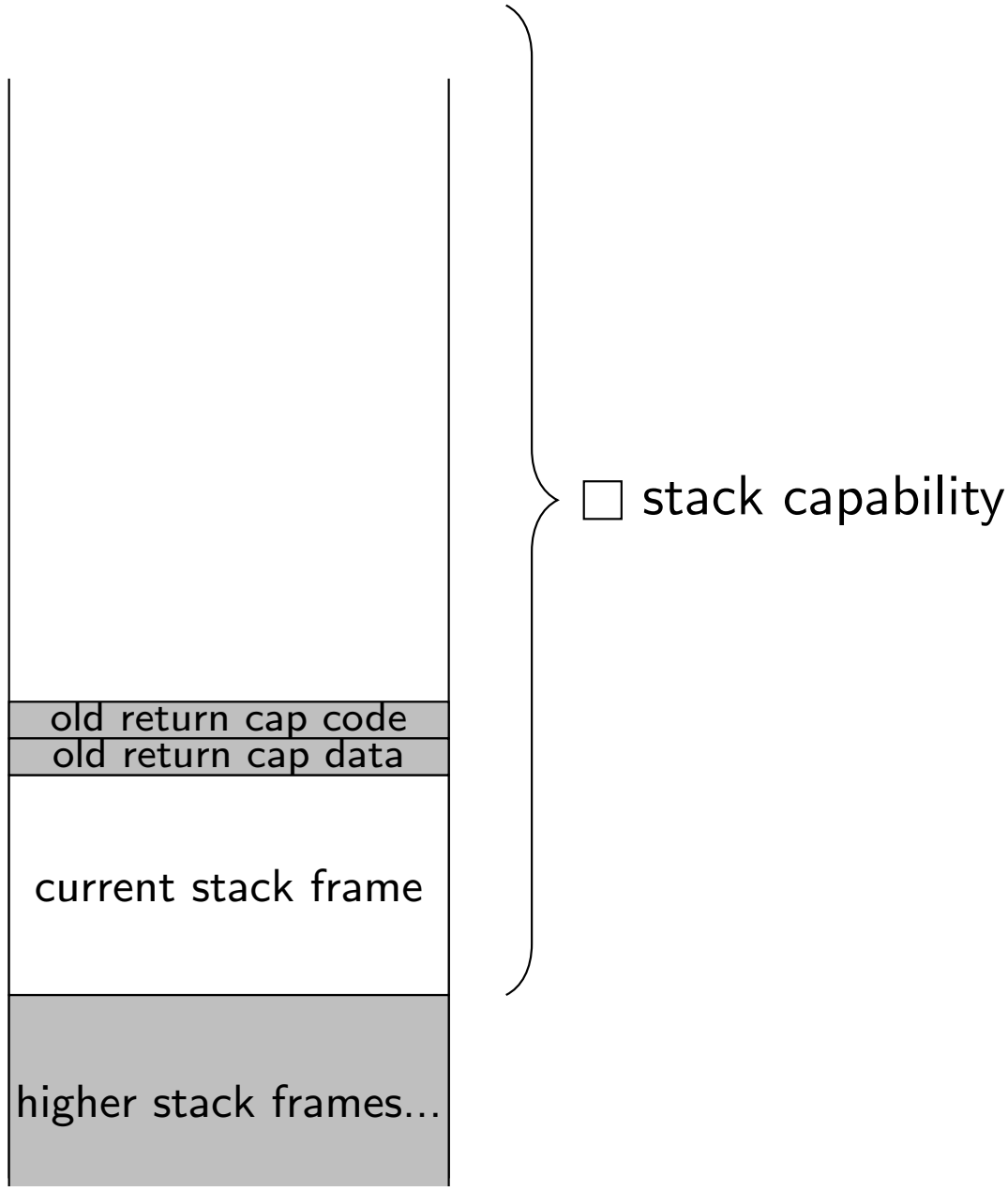
Stack and return capabilities



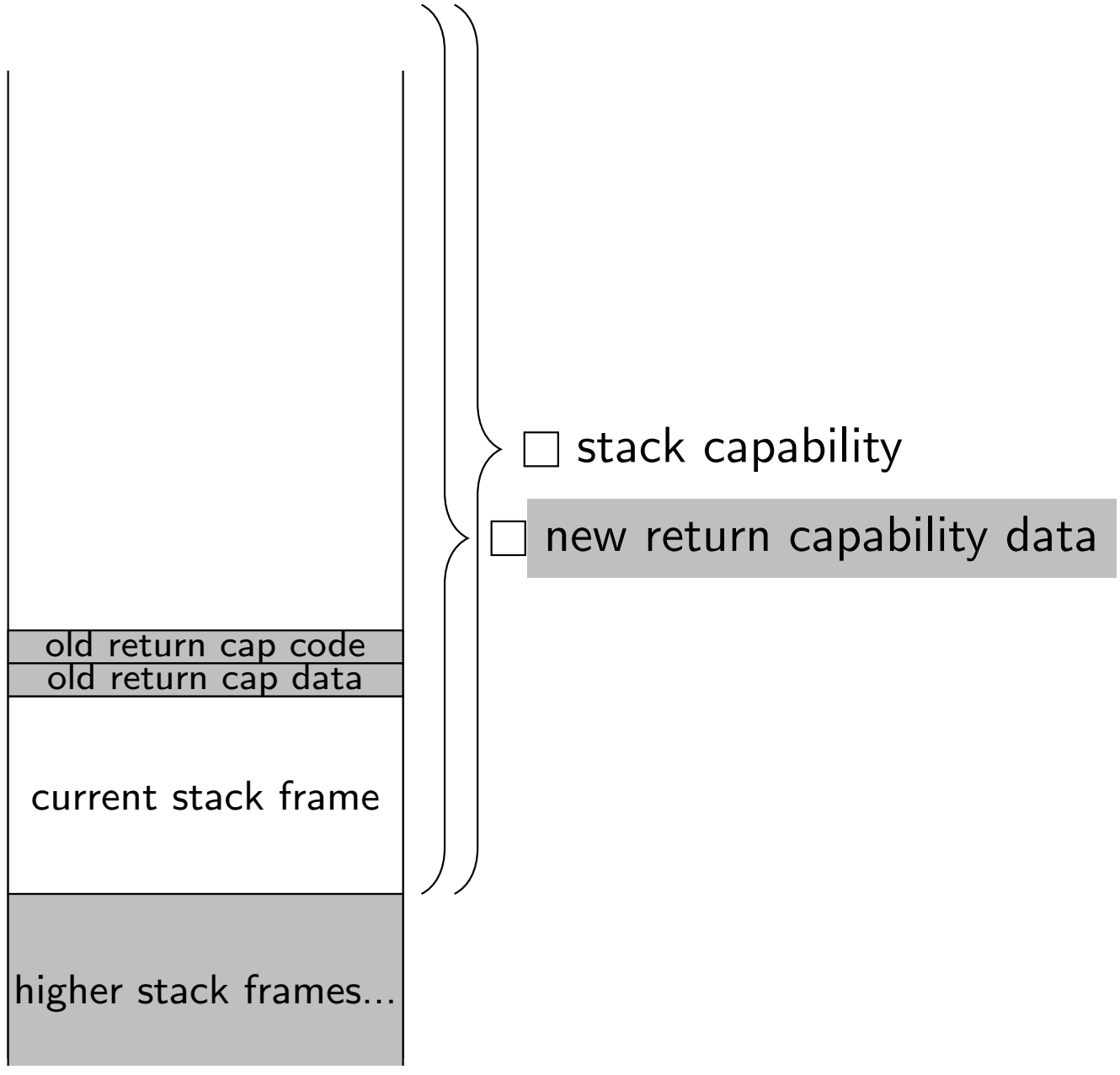
Stack and return capabilities



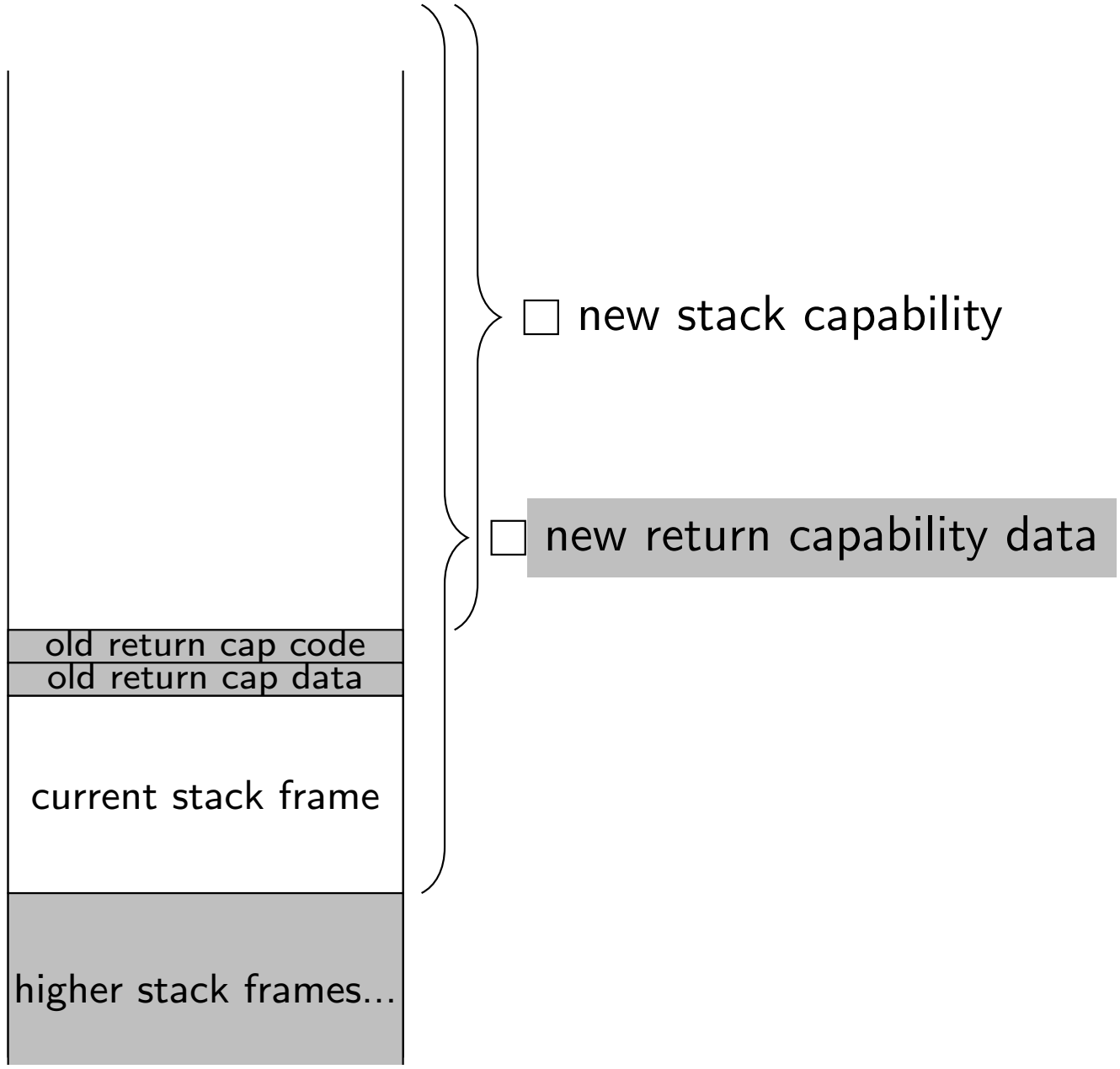
Stack and return capabilities



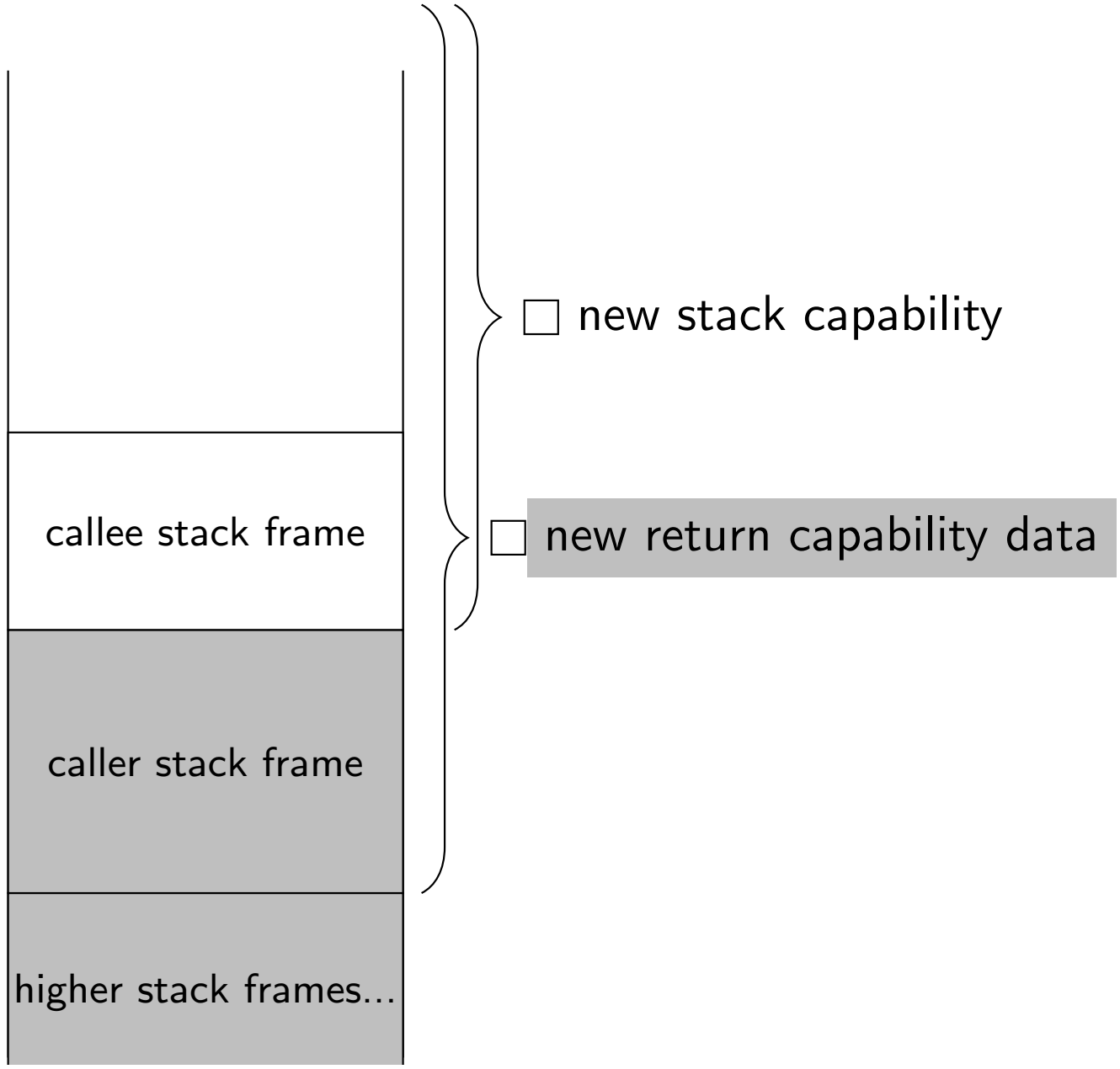
Stack and return capabilities



Stack and return capabilities



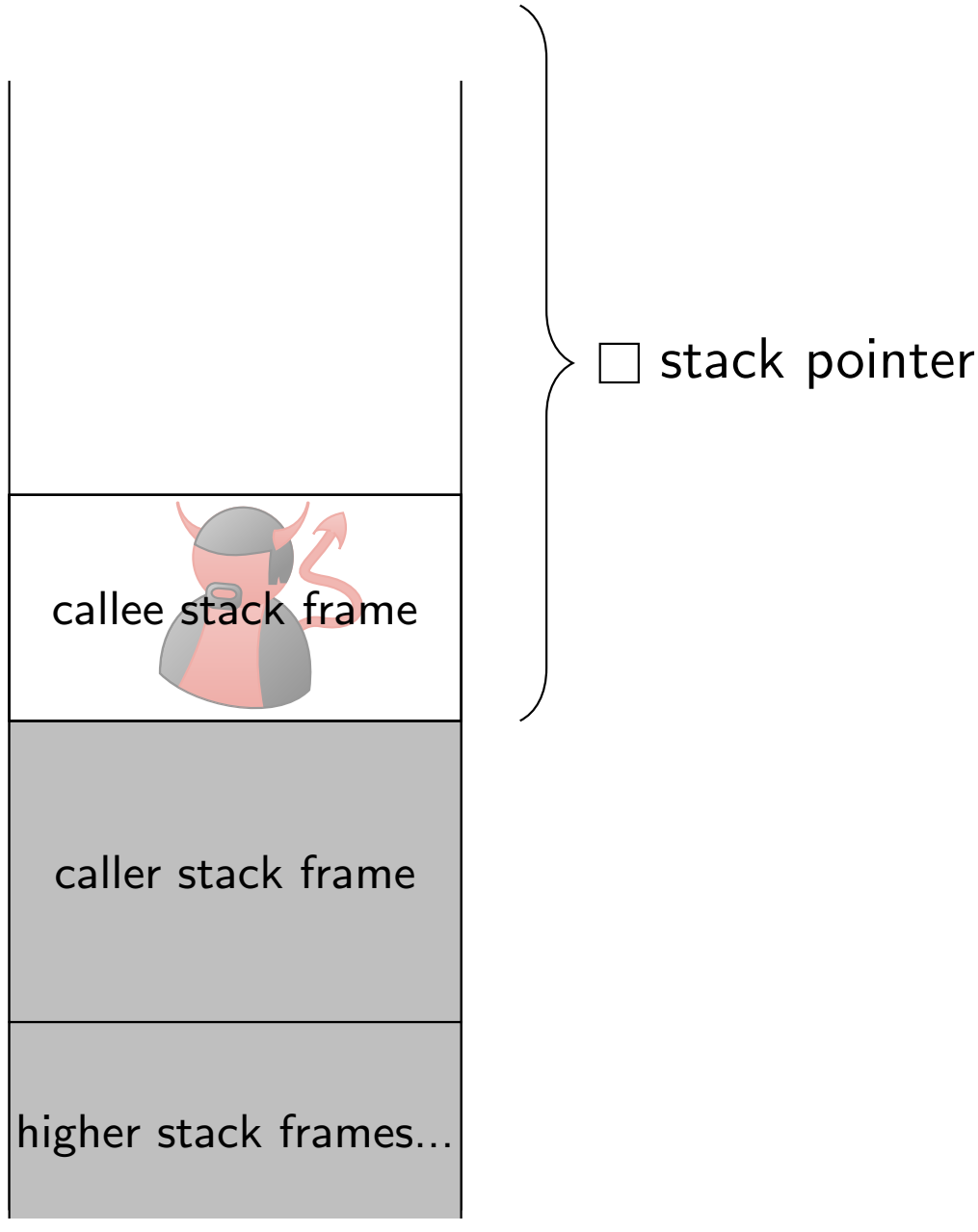
Stack and return capabilities



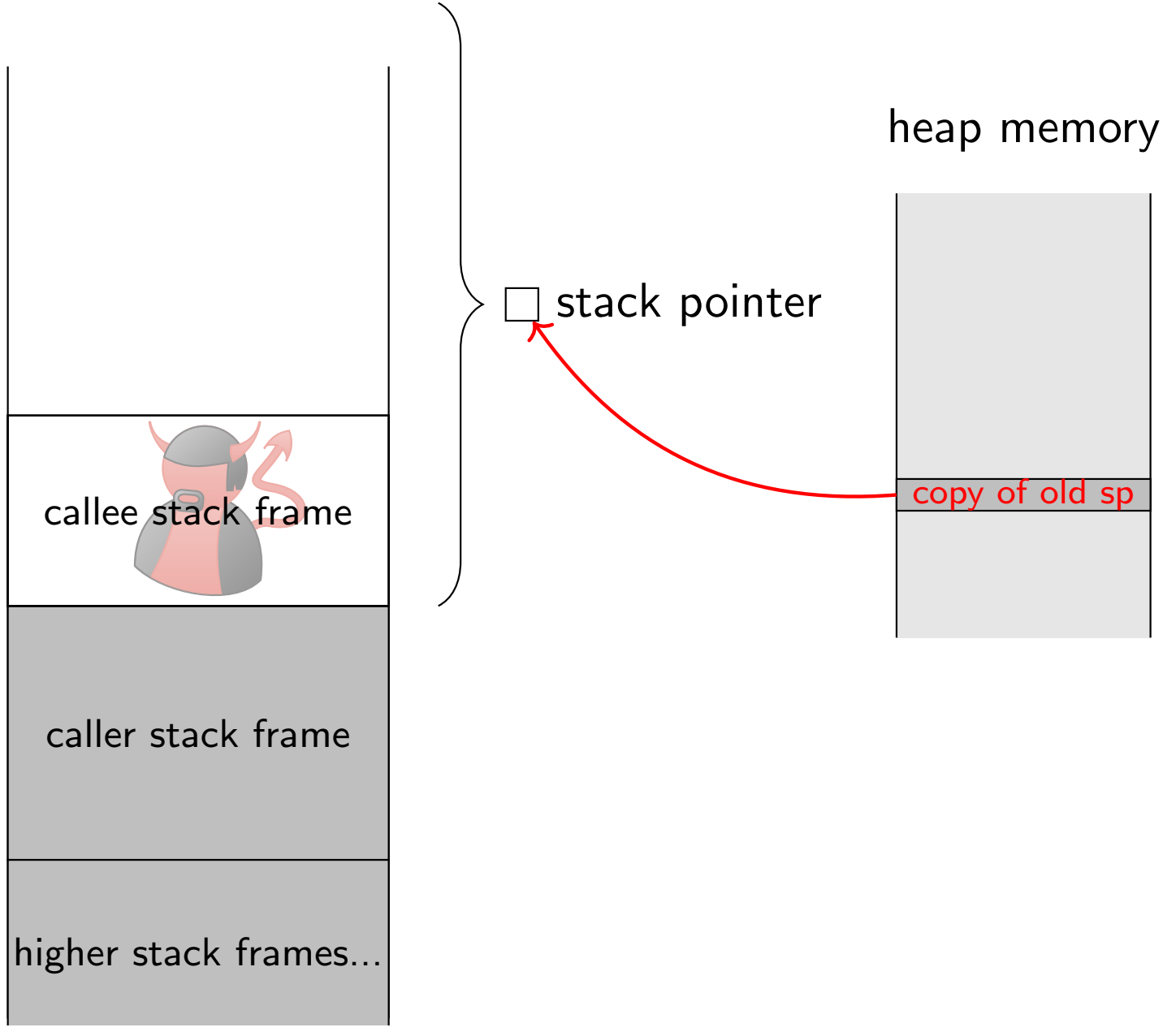
Stack and return pointer security

- Desirable properties:
  - Well-bracketed control flow
  - Stack frame encapsulation
- How to enforce?
  - Capabilities?
    - Not enough: three attacks!

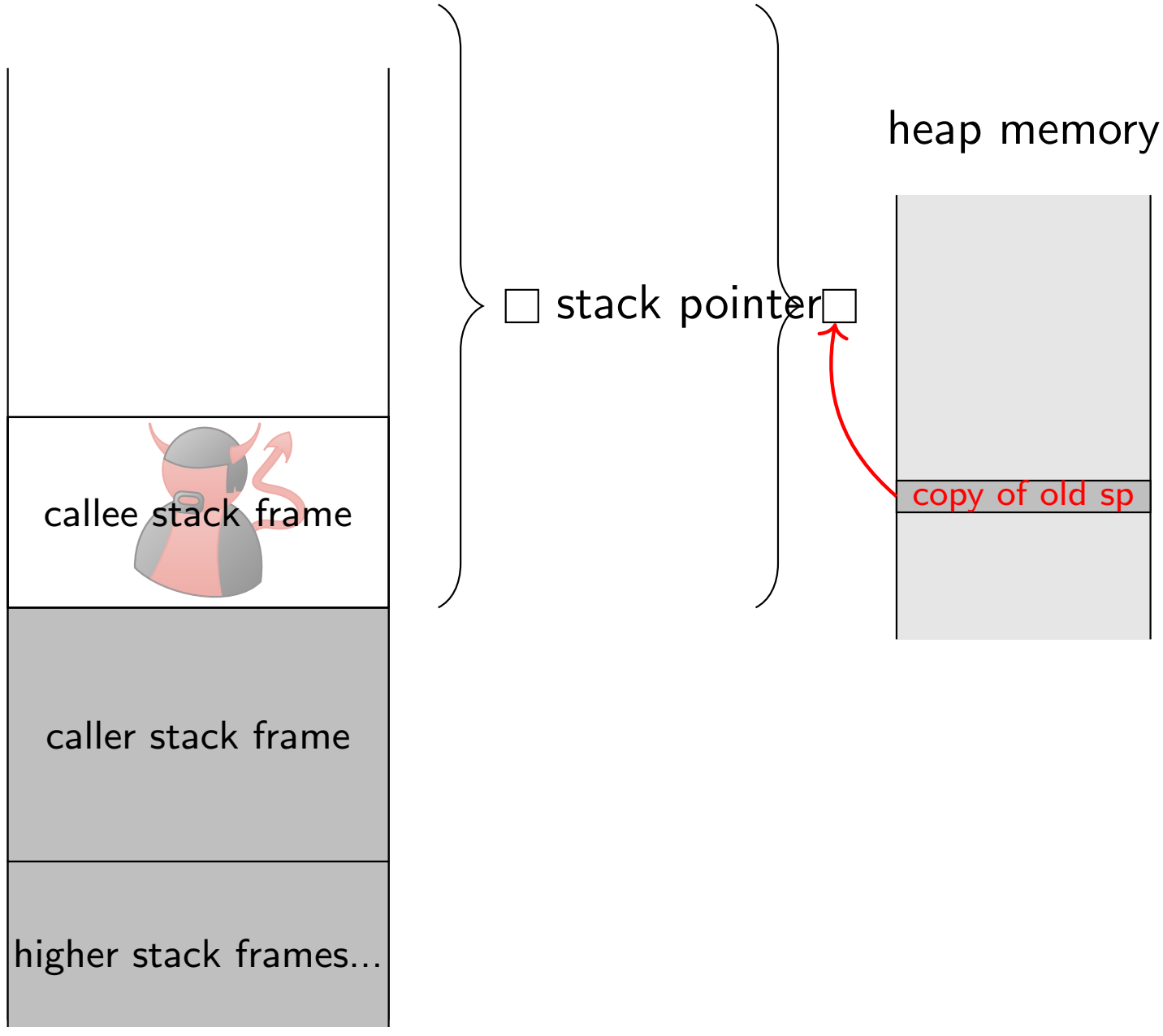
Stack and return capabilities: Attack 1



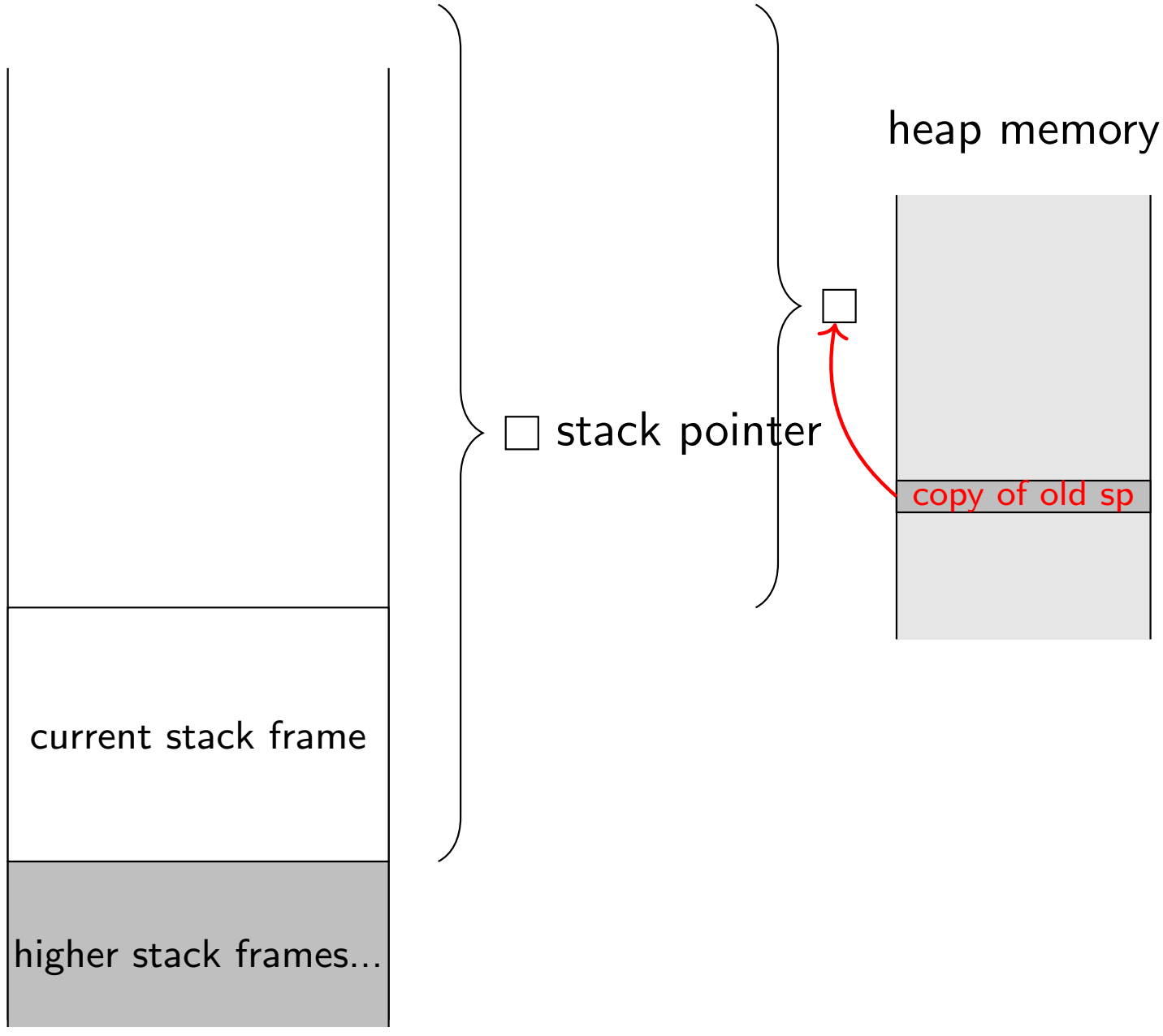
Stack and return capabilities: Attack 1



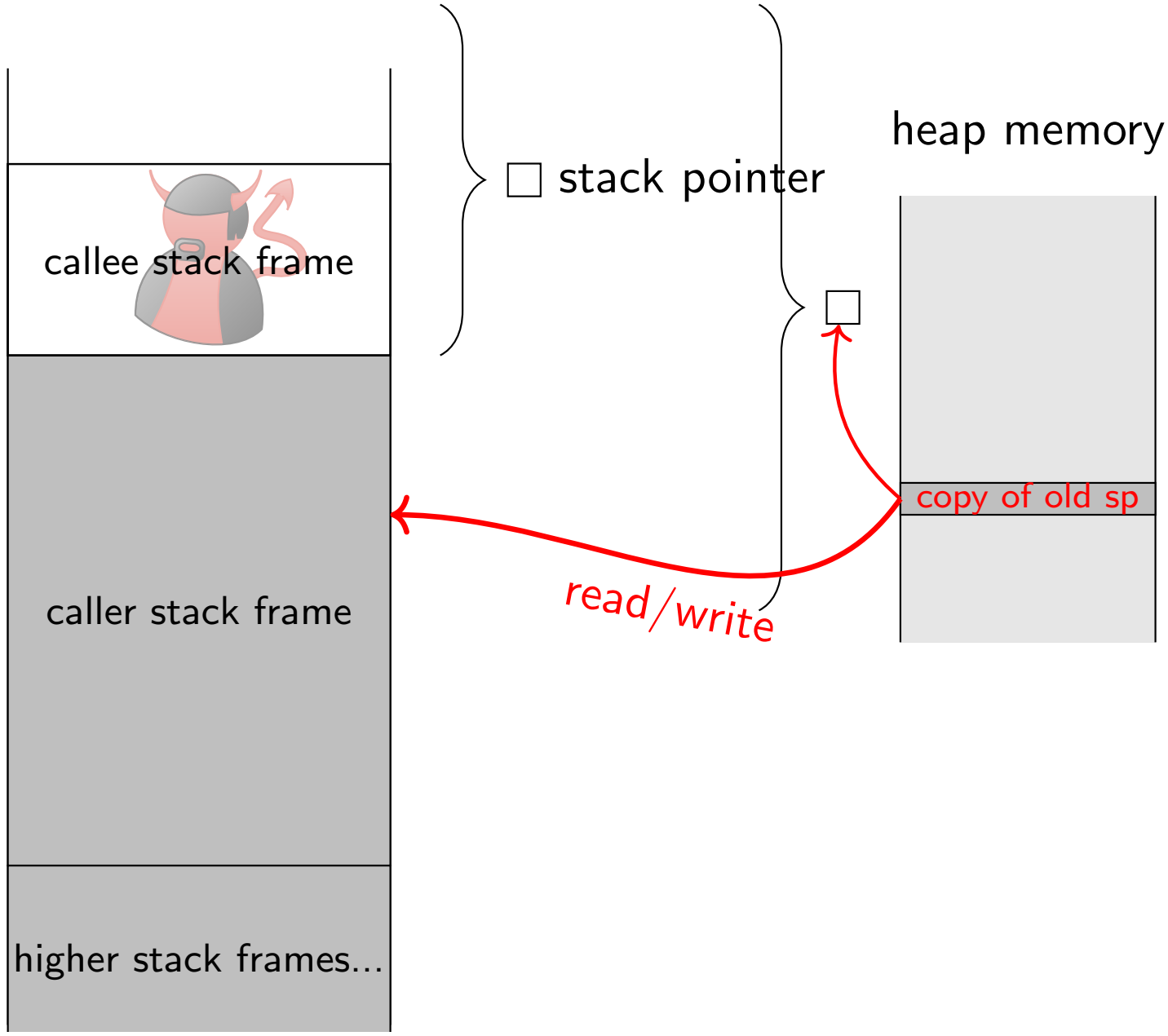
Stack and return capabilities: Attack 1



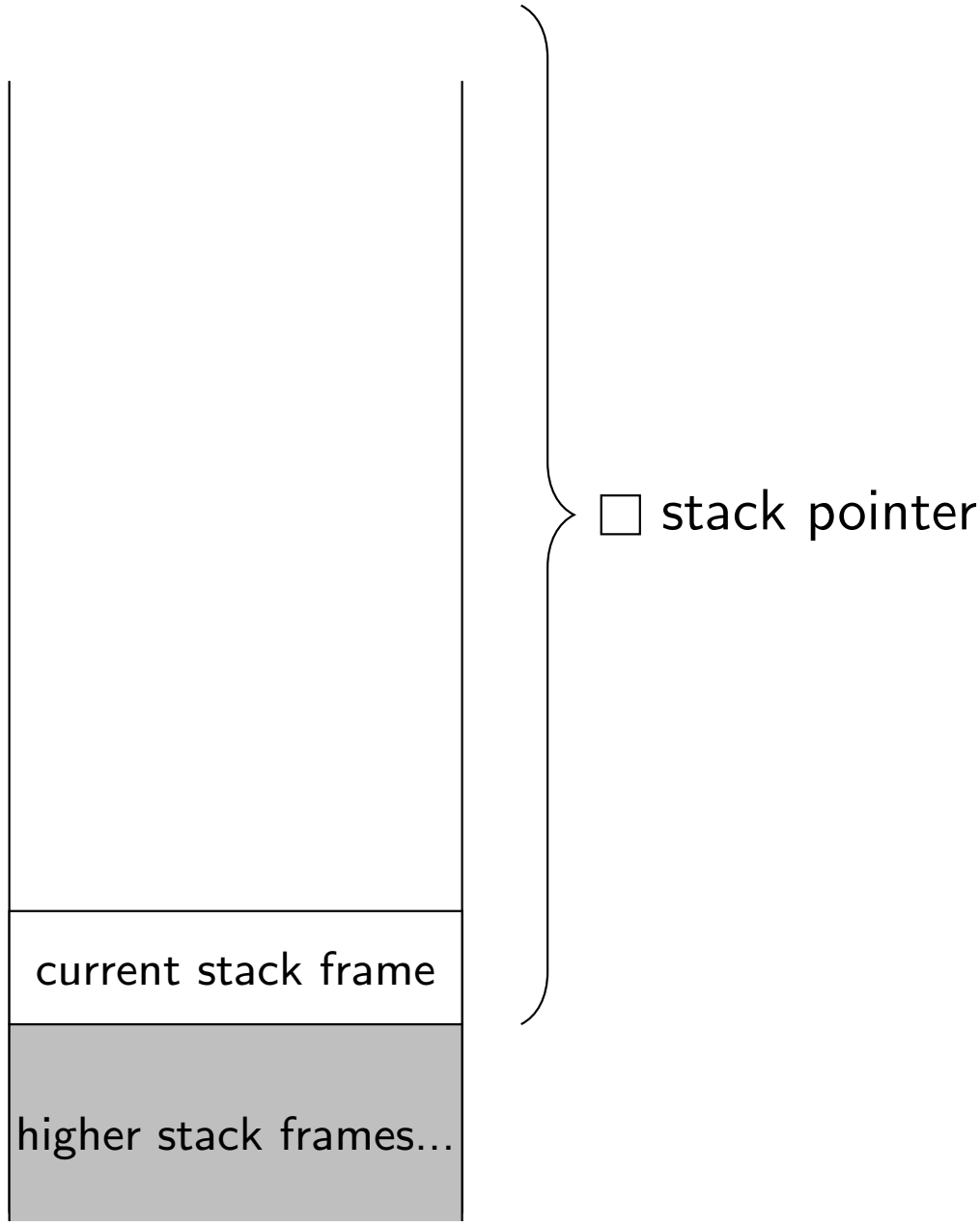
Stack and return capabilities: Attack 1



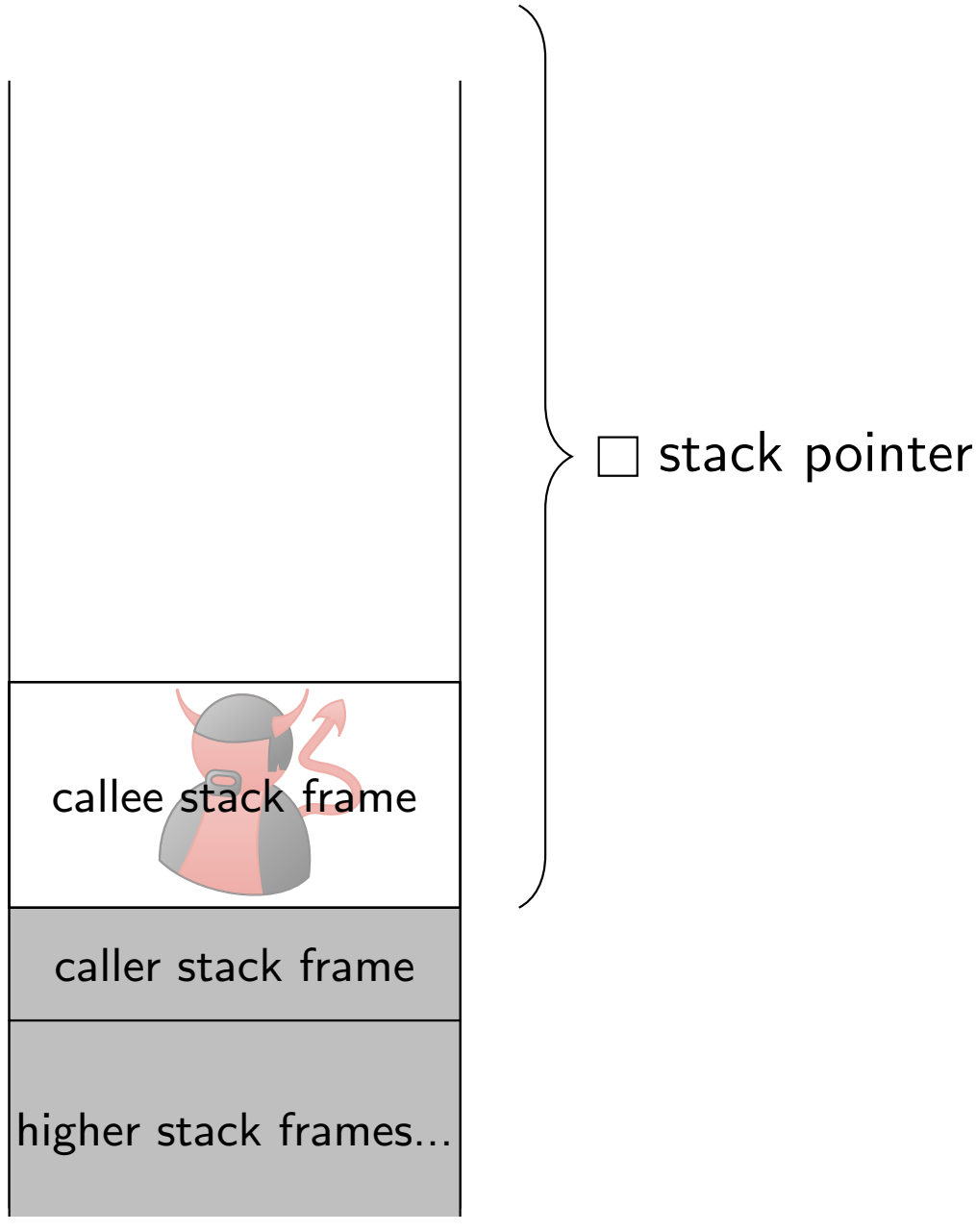
Stack and return capabilities: Attack 1



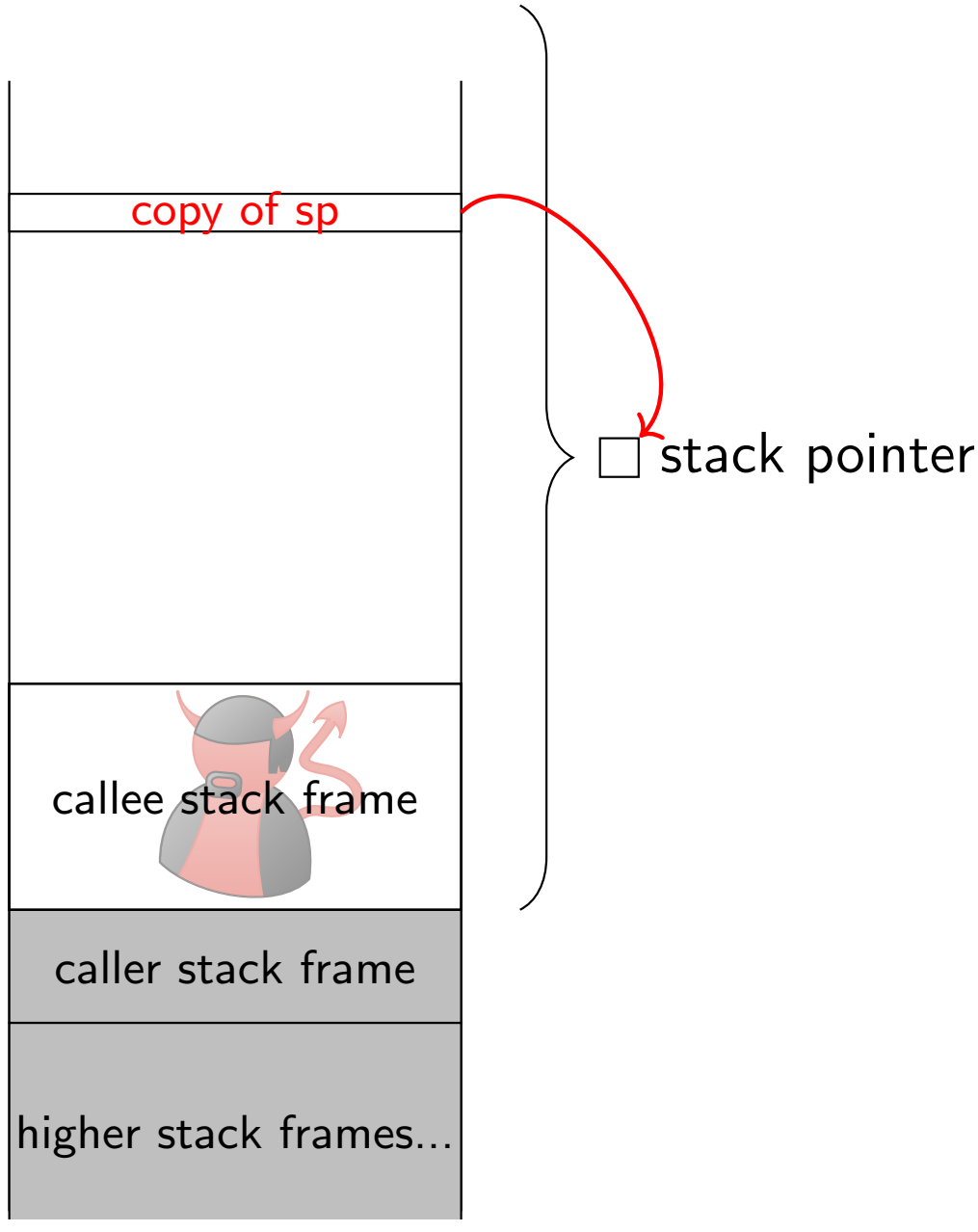
Stack and return capabilities: Attack 2



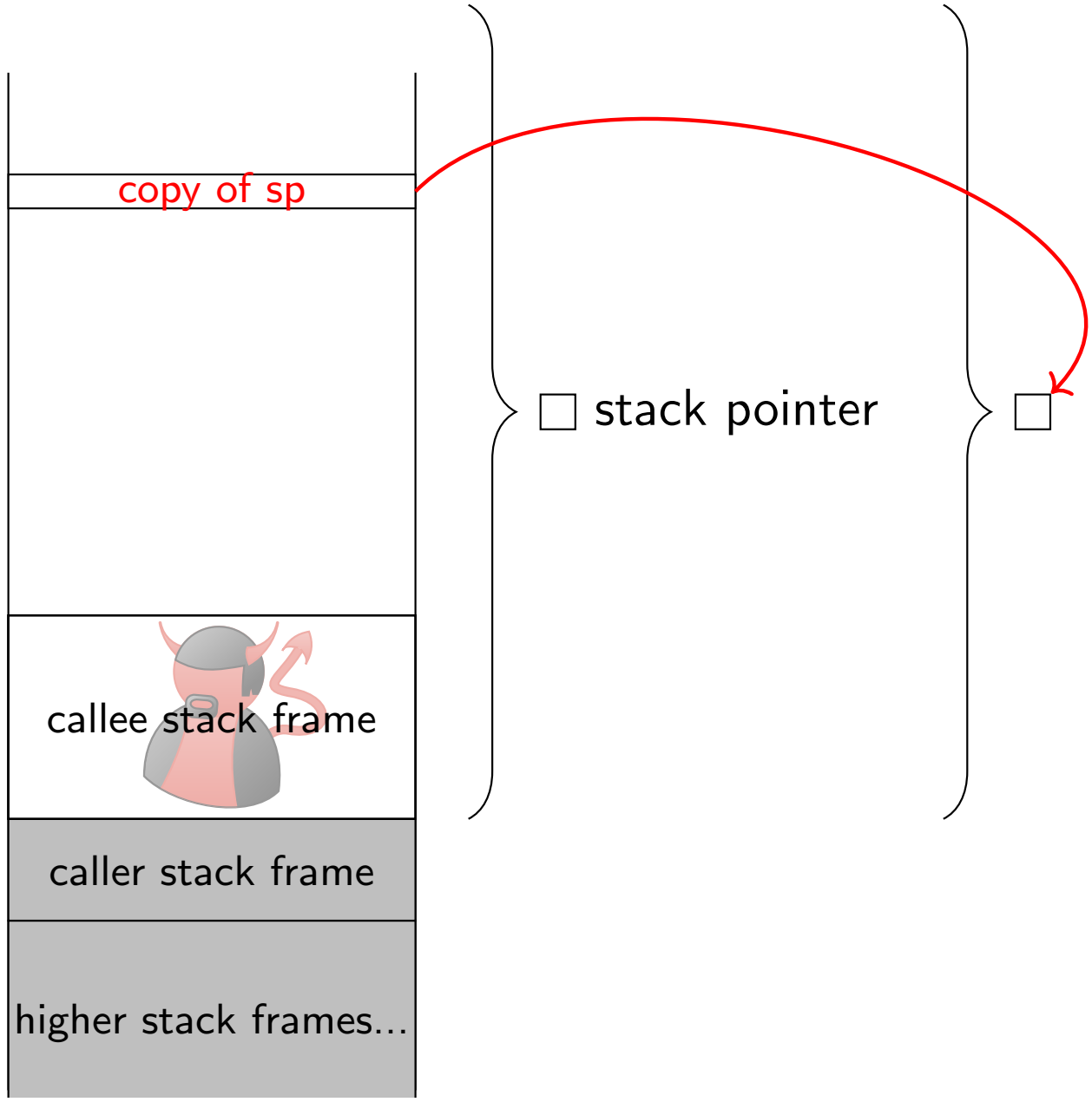
Stack and return capabilities: Attack 2



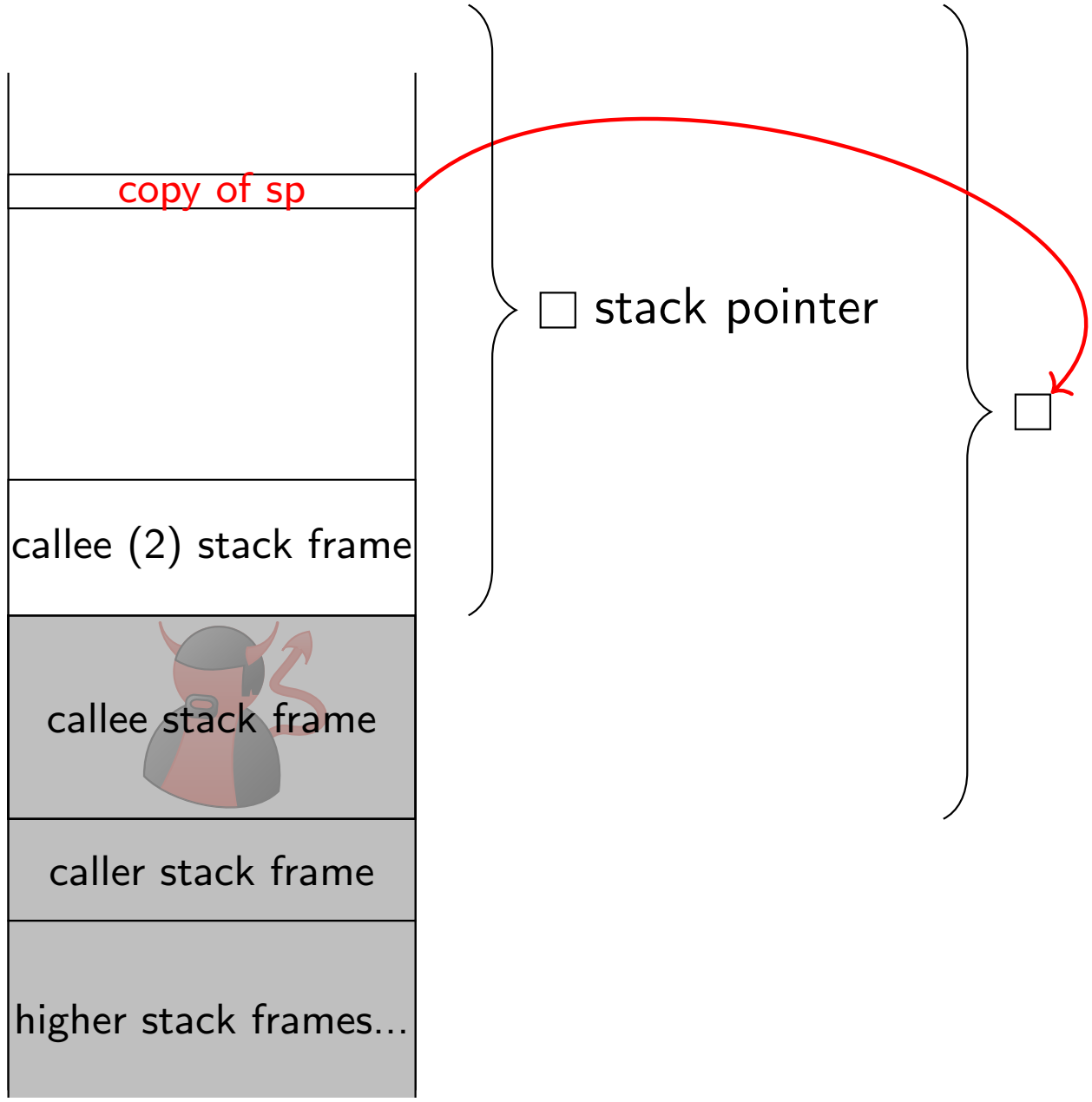
Stack and return capabilities: Attack 2



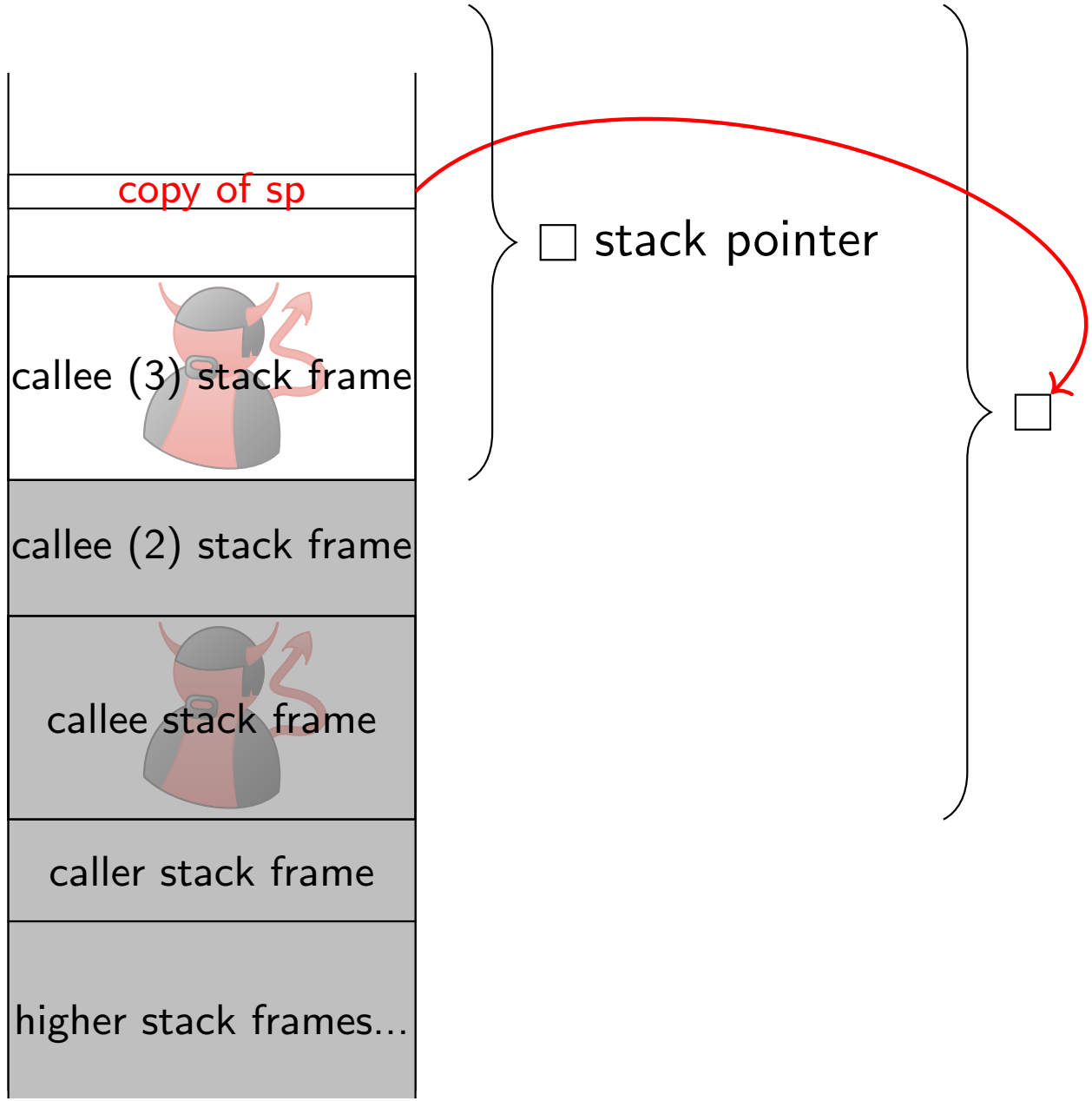
Stack and return capabilities: Attack 2



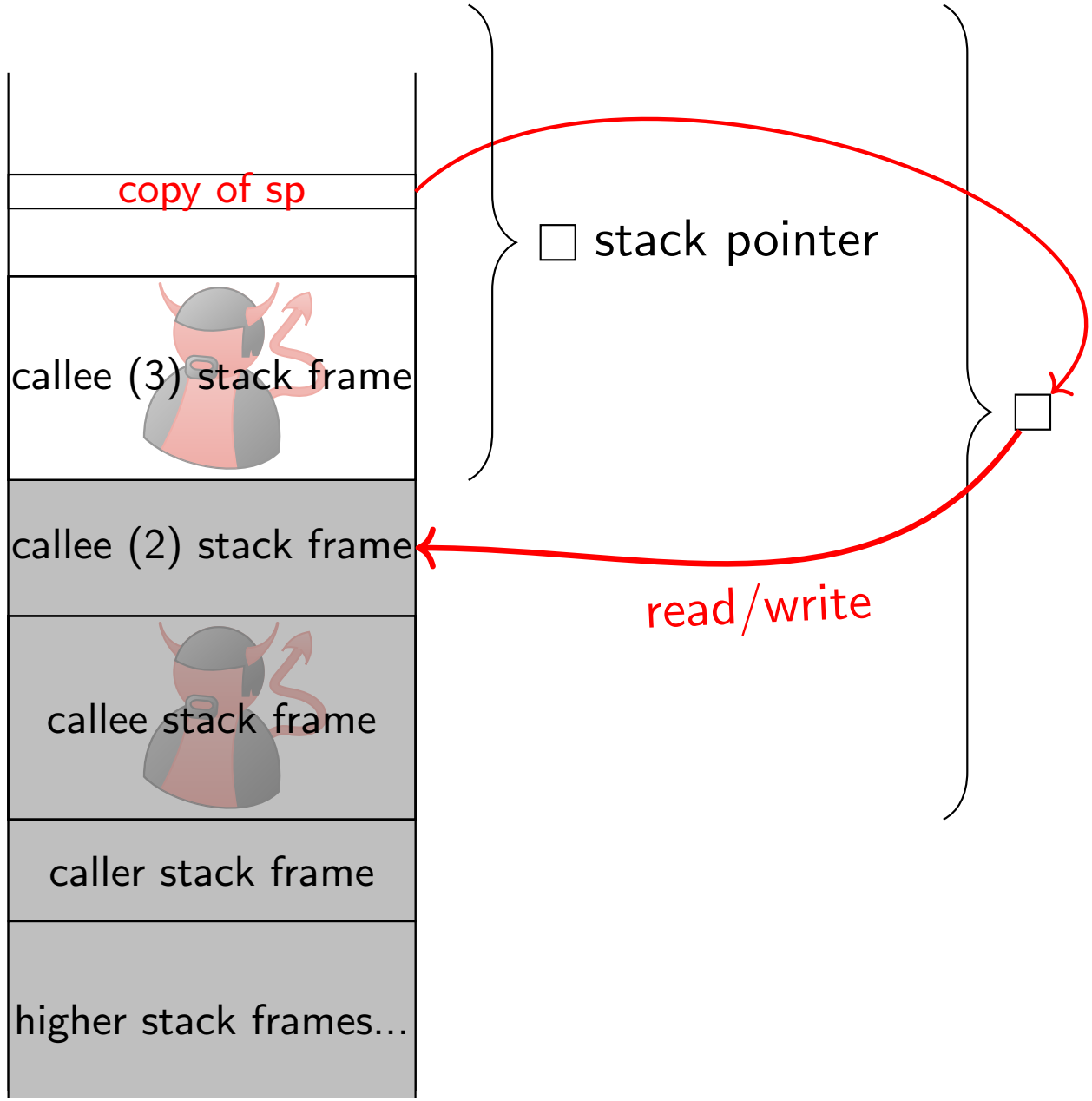
Stack and return capabilities: Attack 2



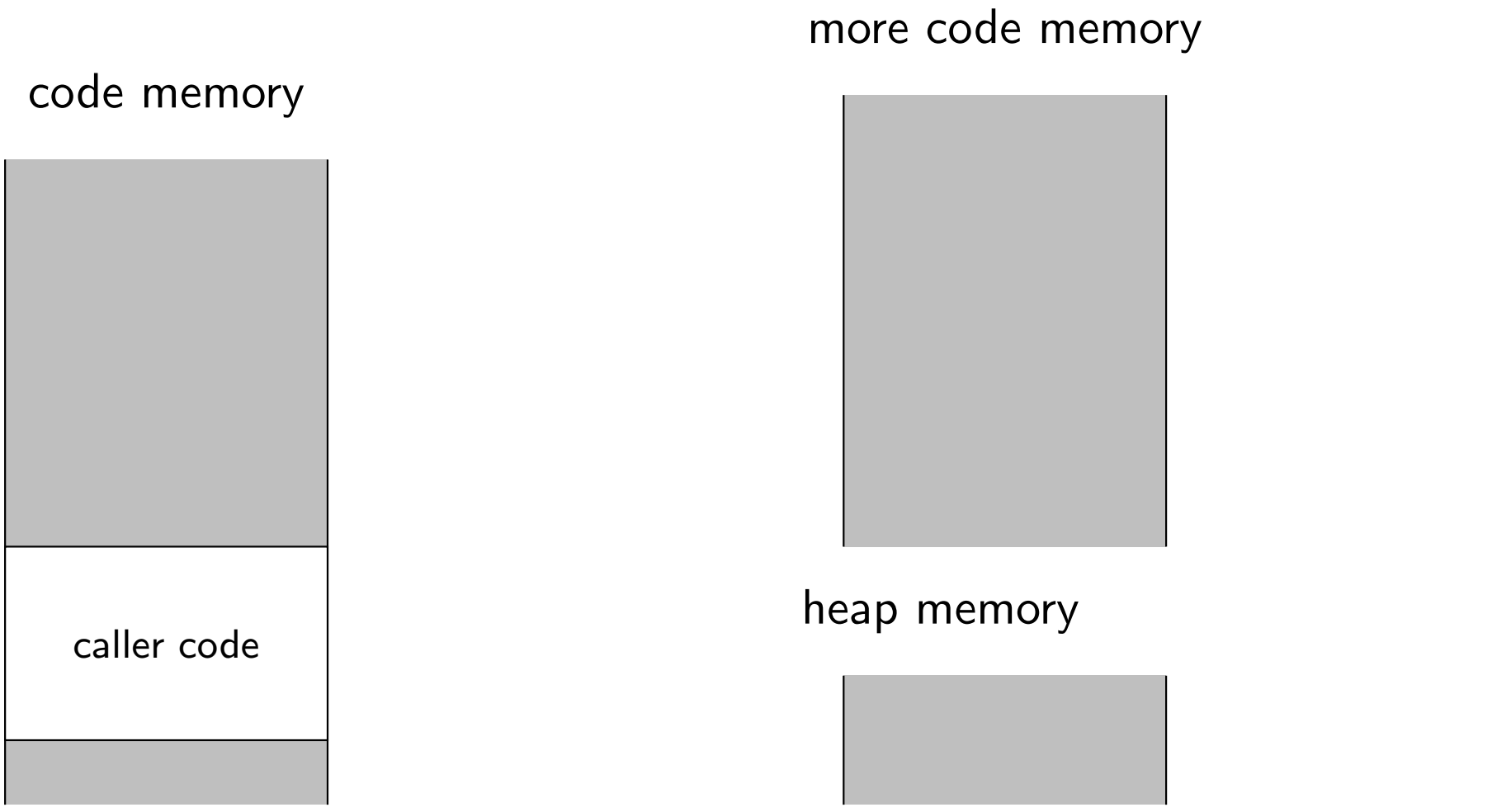
Stack and return capabilities: Attack 2



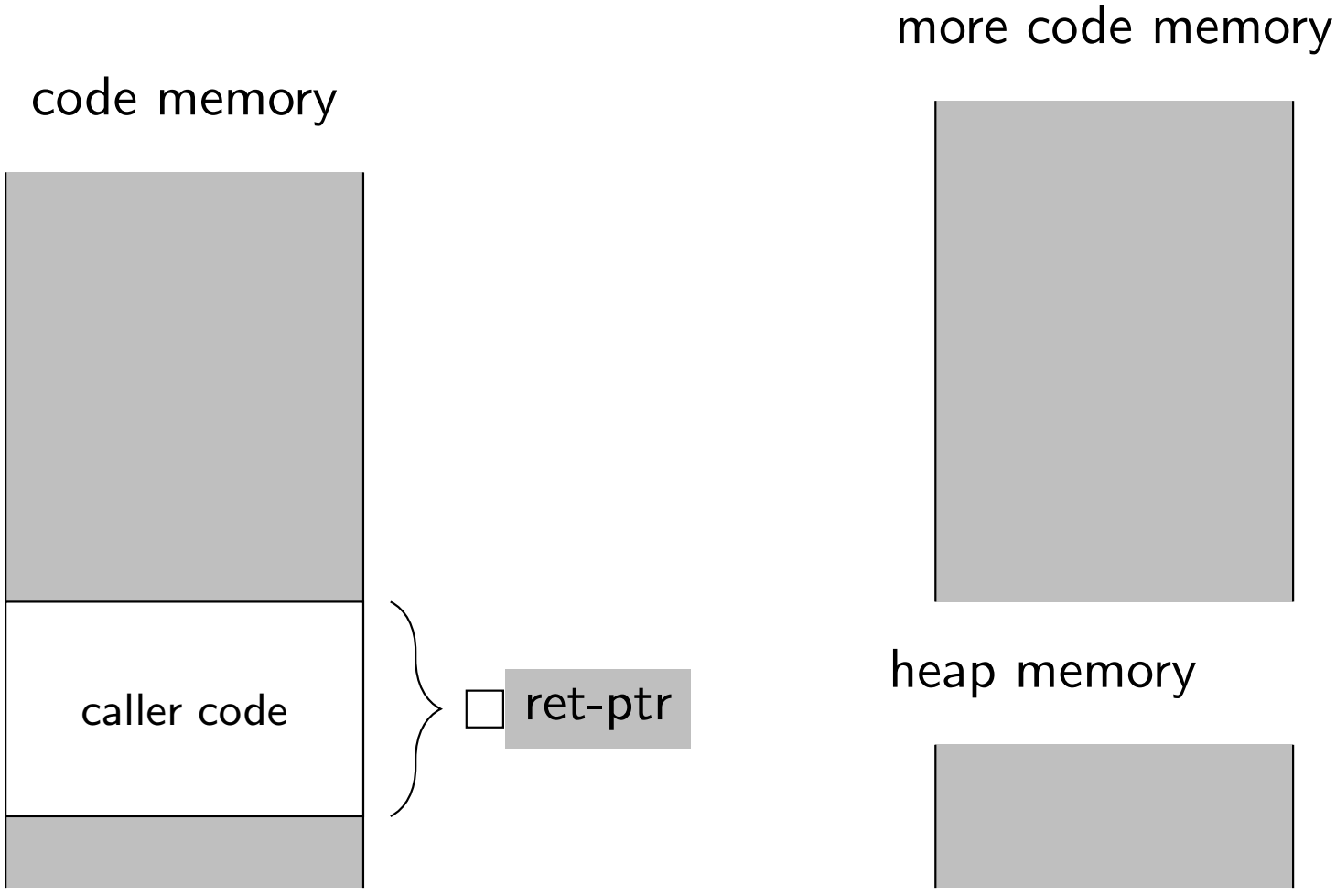
Stack and return capabilities: Attack 2



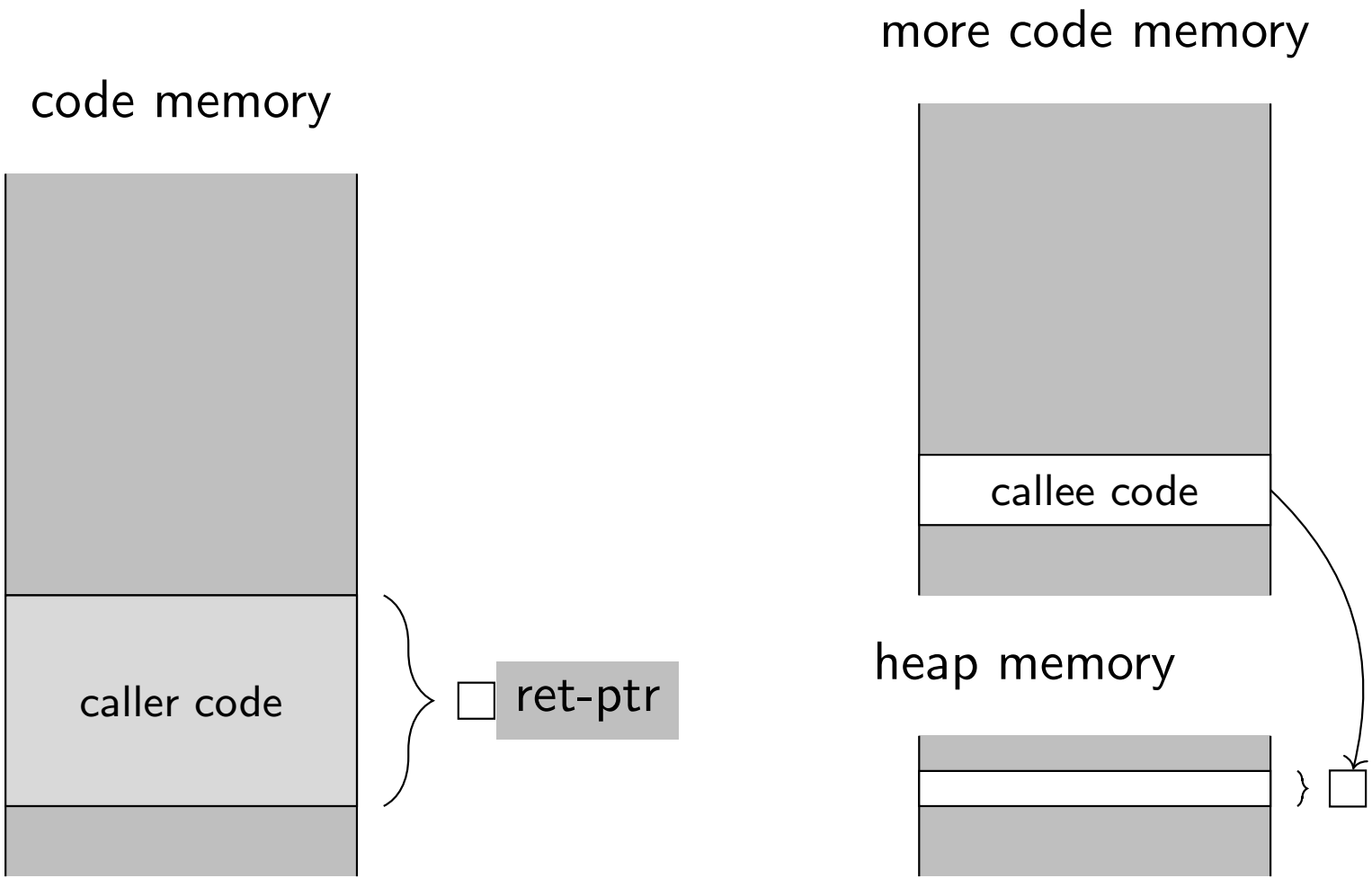
Stack and return capabilities: Attack 3



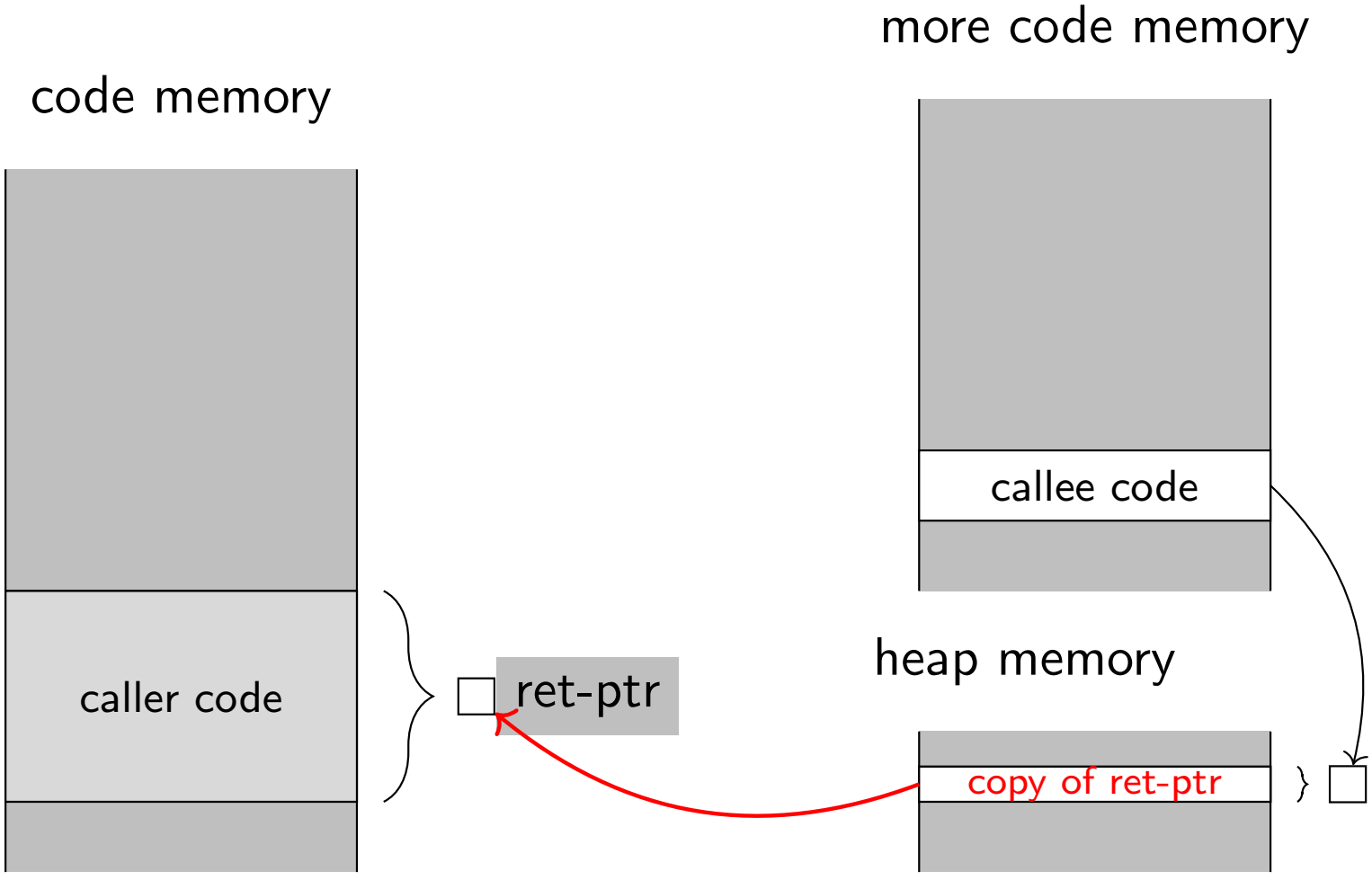
Stack and return capabilities: Attack 3



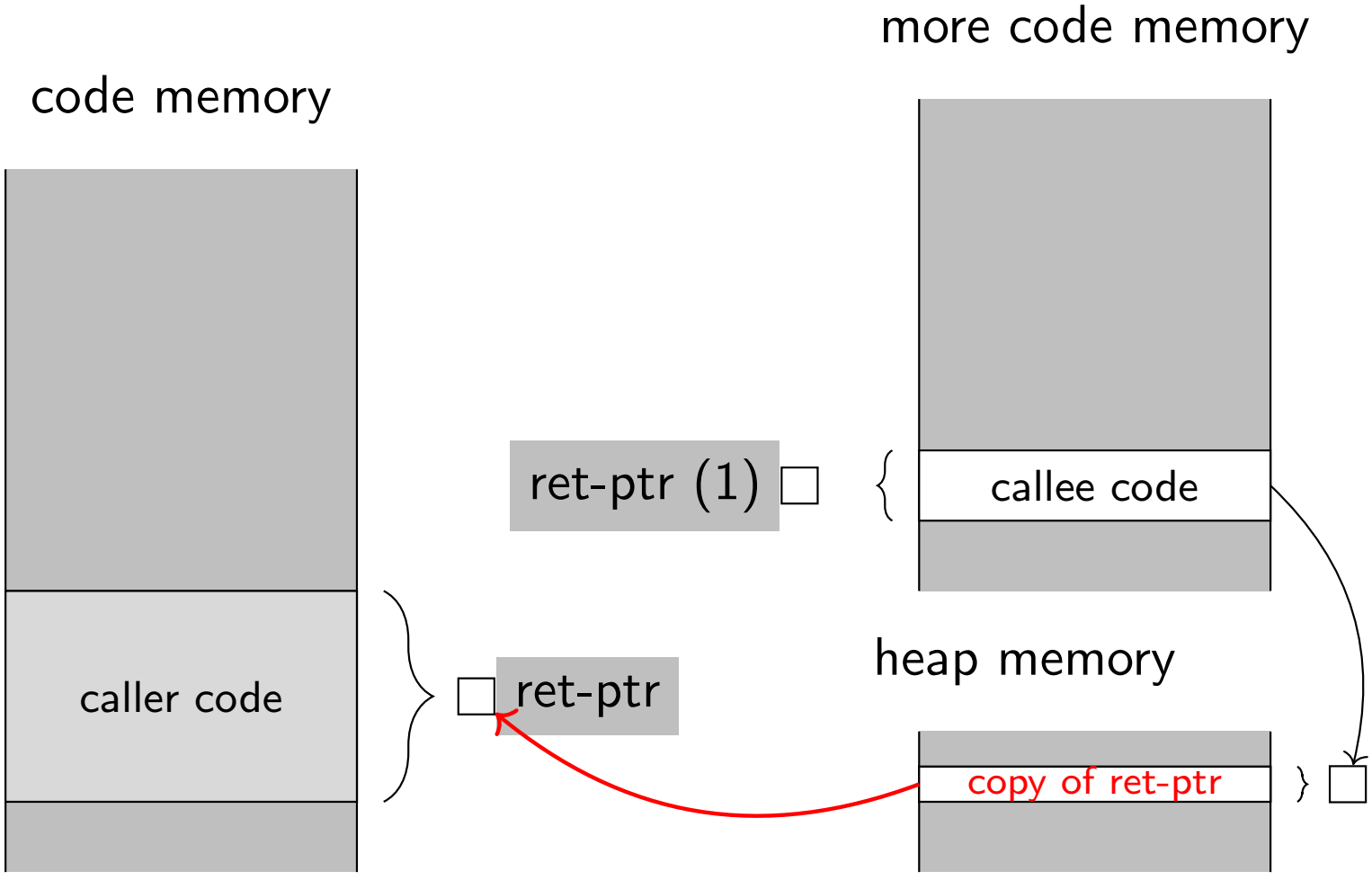
Stack and return capabilities: Attack 3



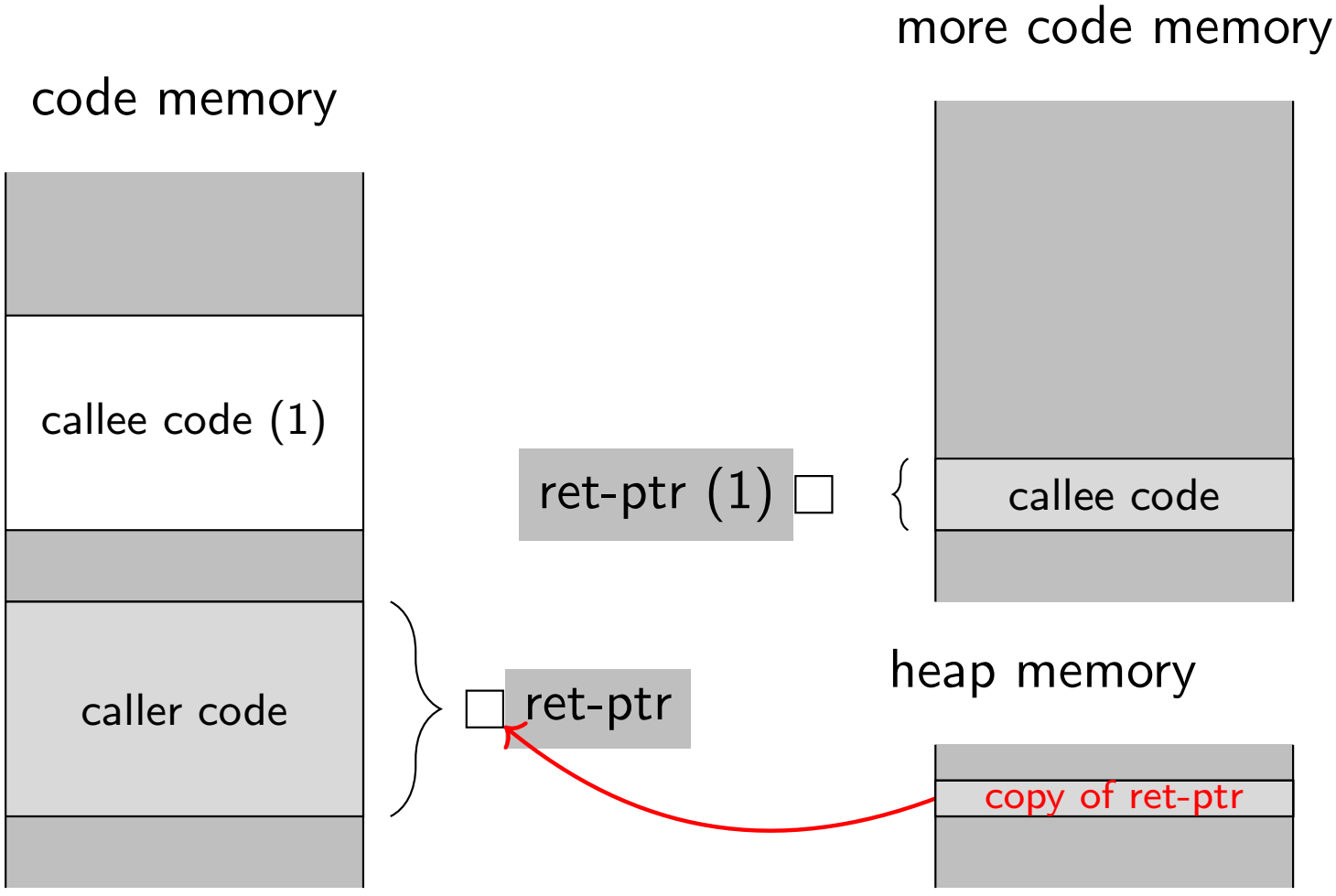
Stack and return capabilities: Attack 3



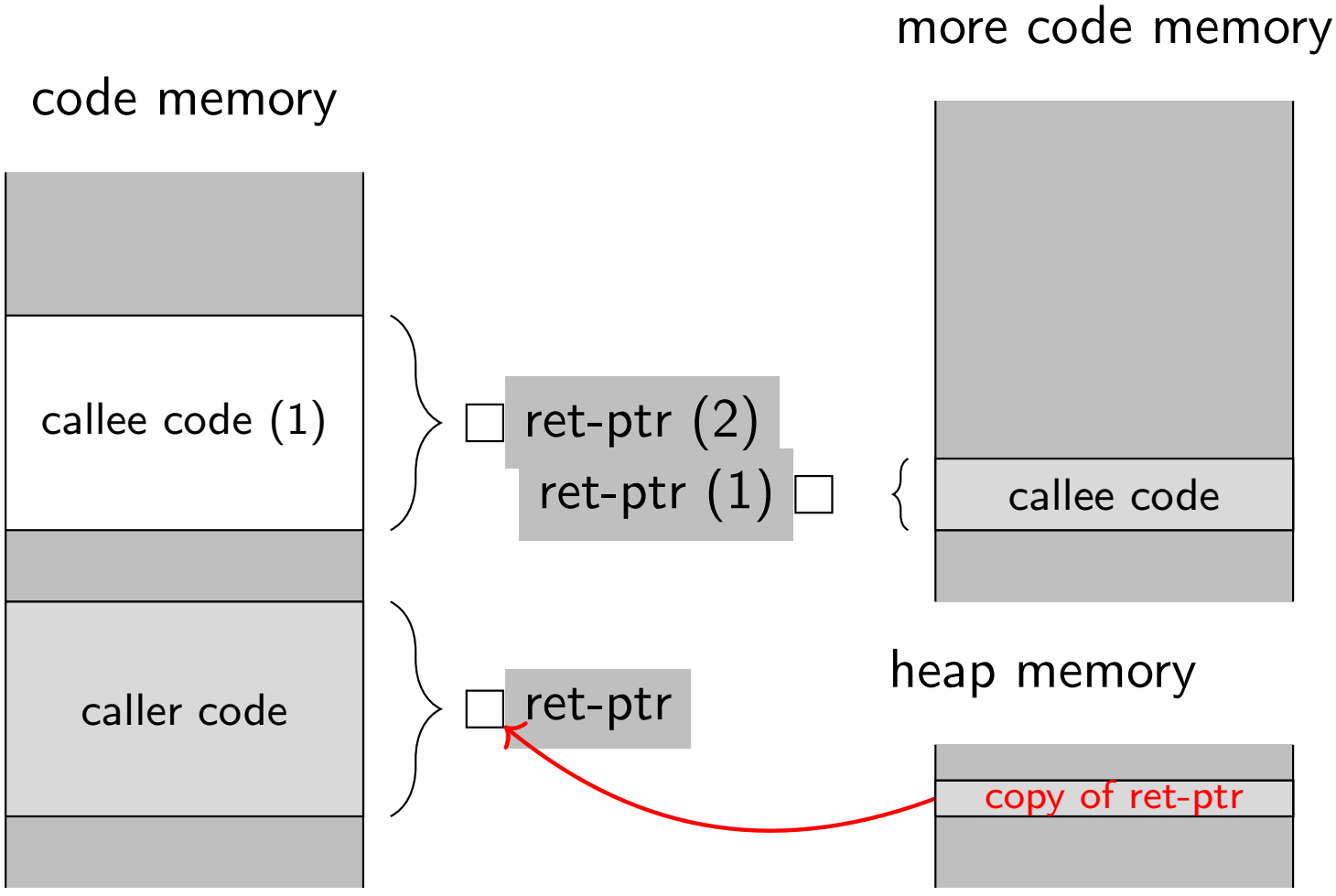
Stack and return capabilities: Attack 3



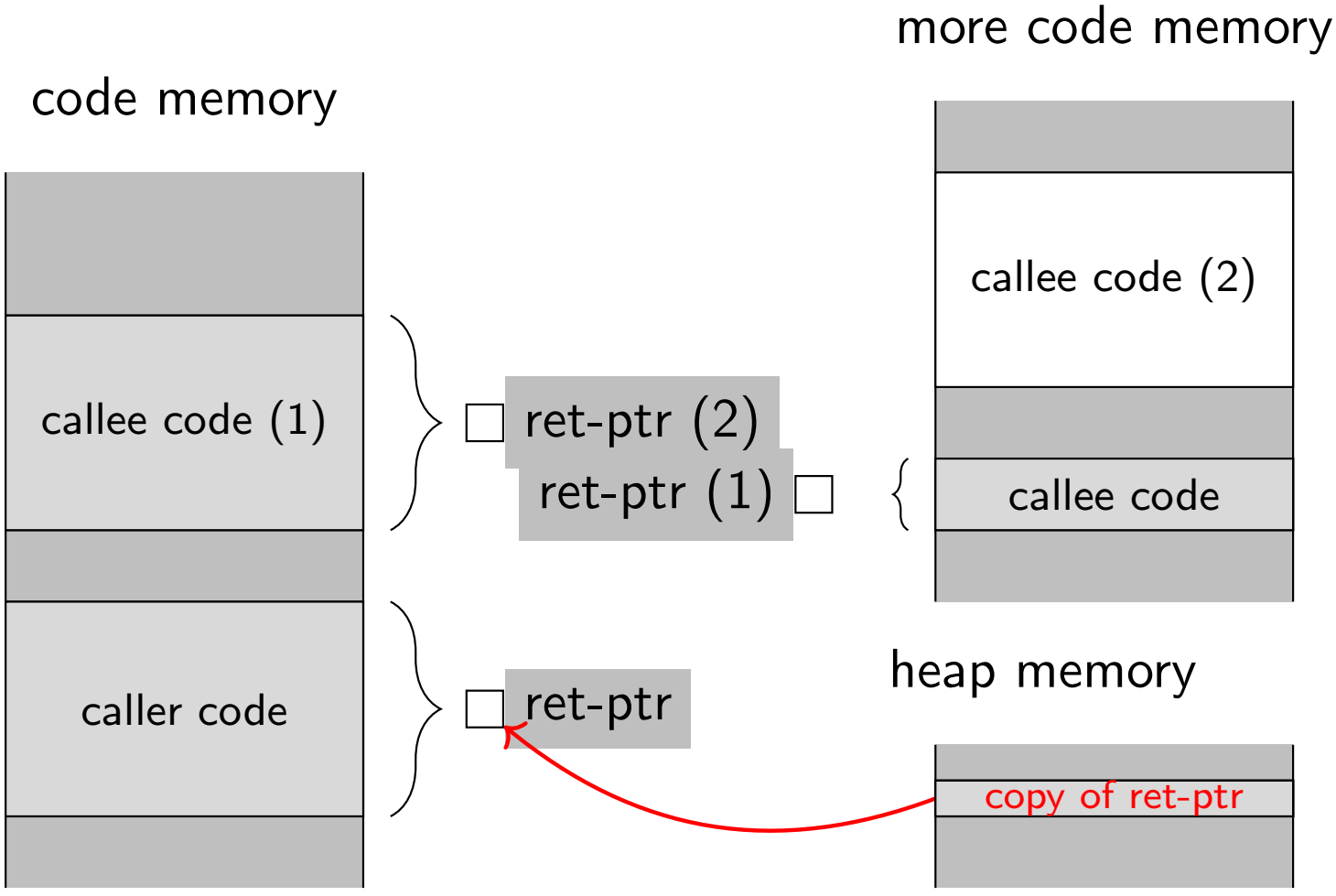
Stack and return capabilities: Attack 3



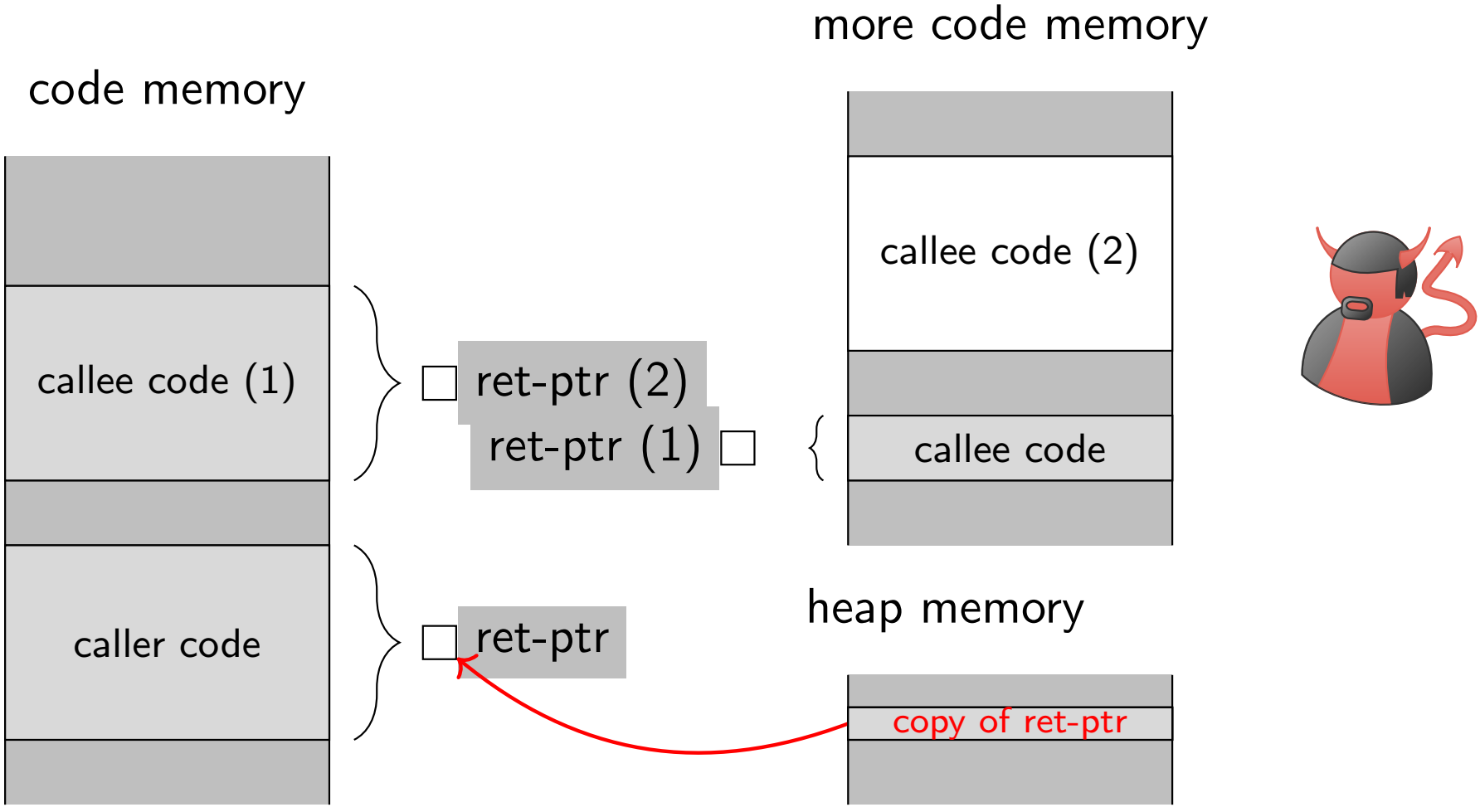
Stack and return capabilities: Attack 3



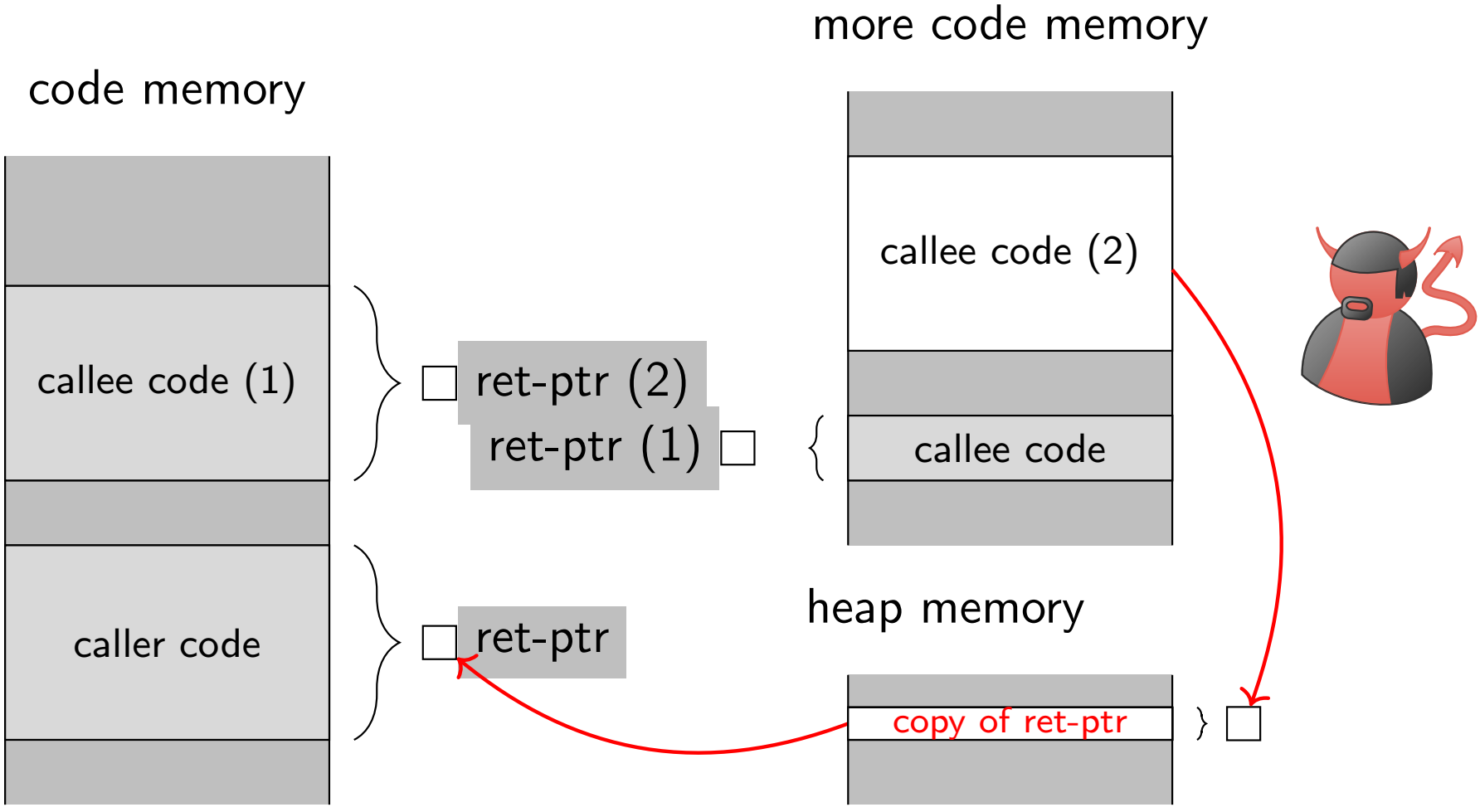
Stack and return capabilities: Attack 3



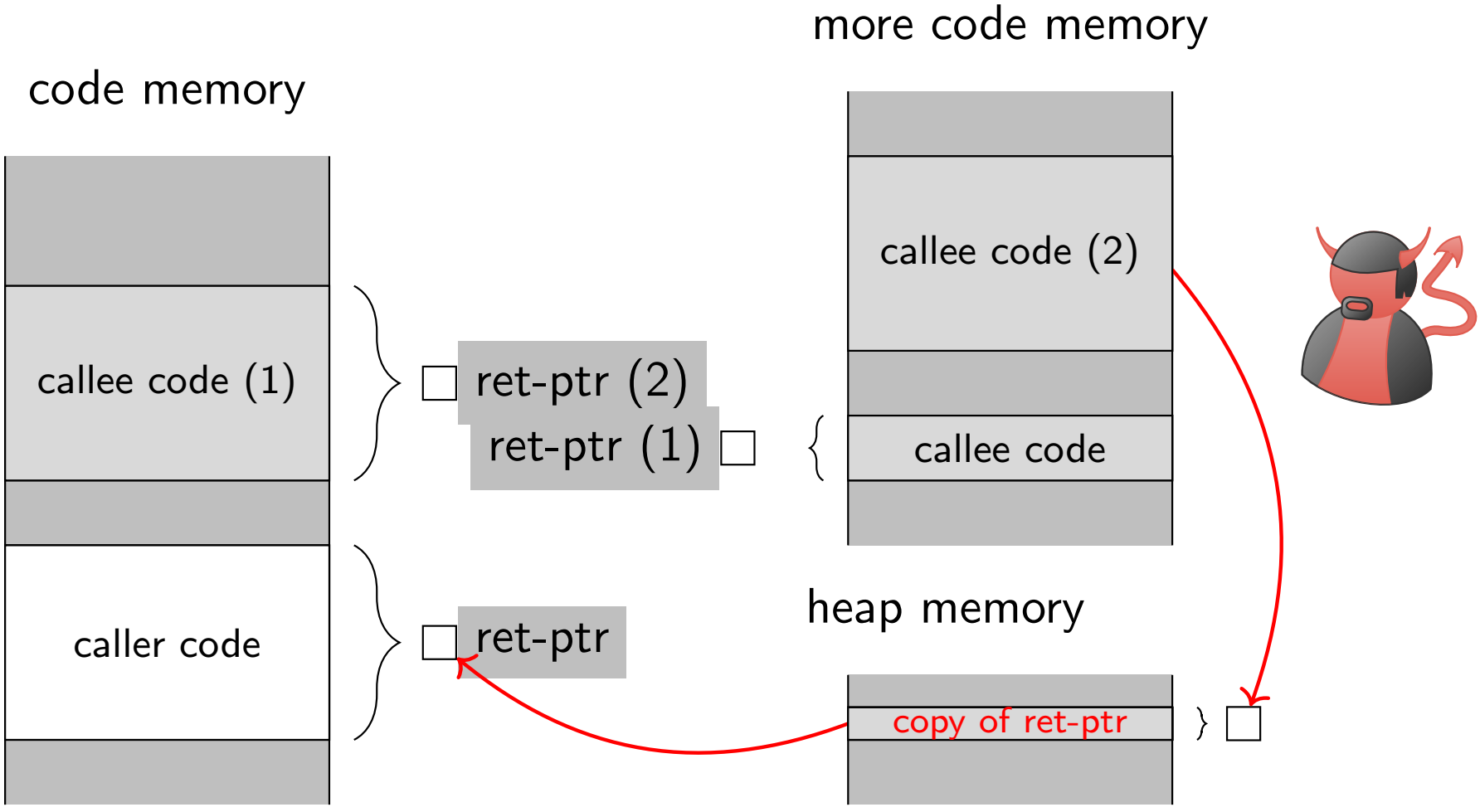
Stack and return capabilities: Attack 3



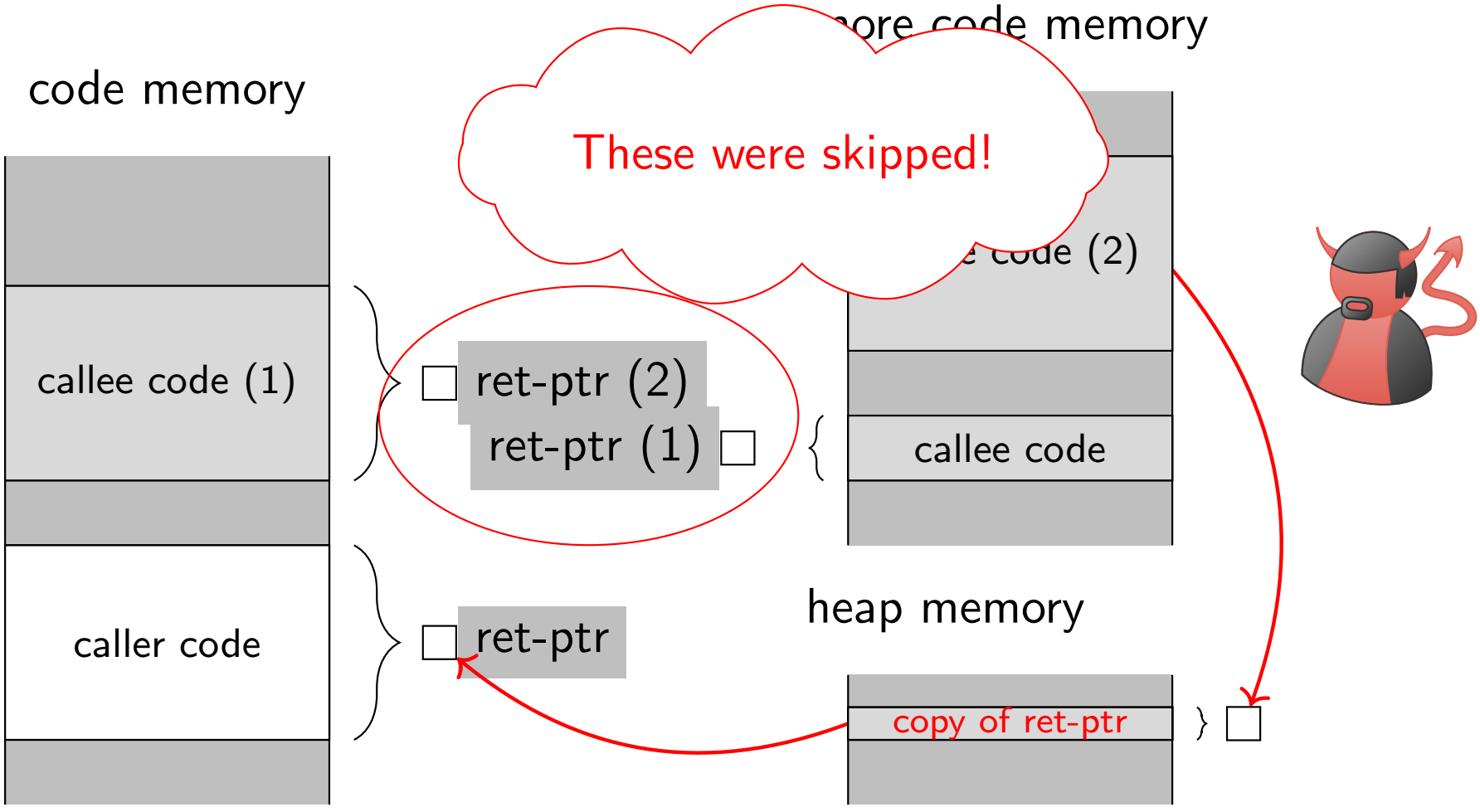
Stack and return capabilities: Attack 3



Stack and return capabilities: Attack 3



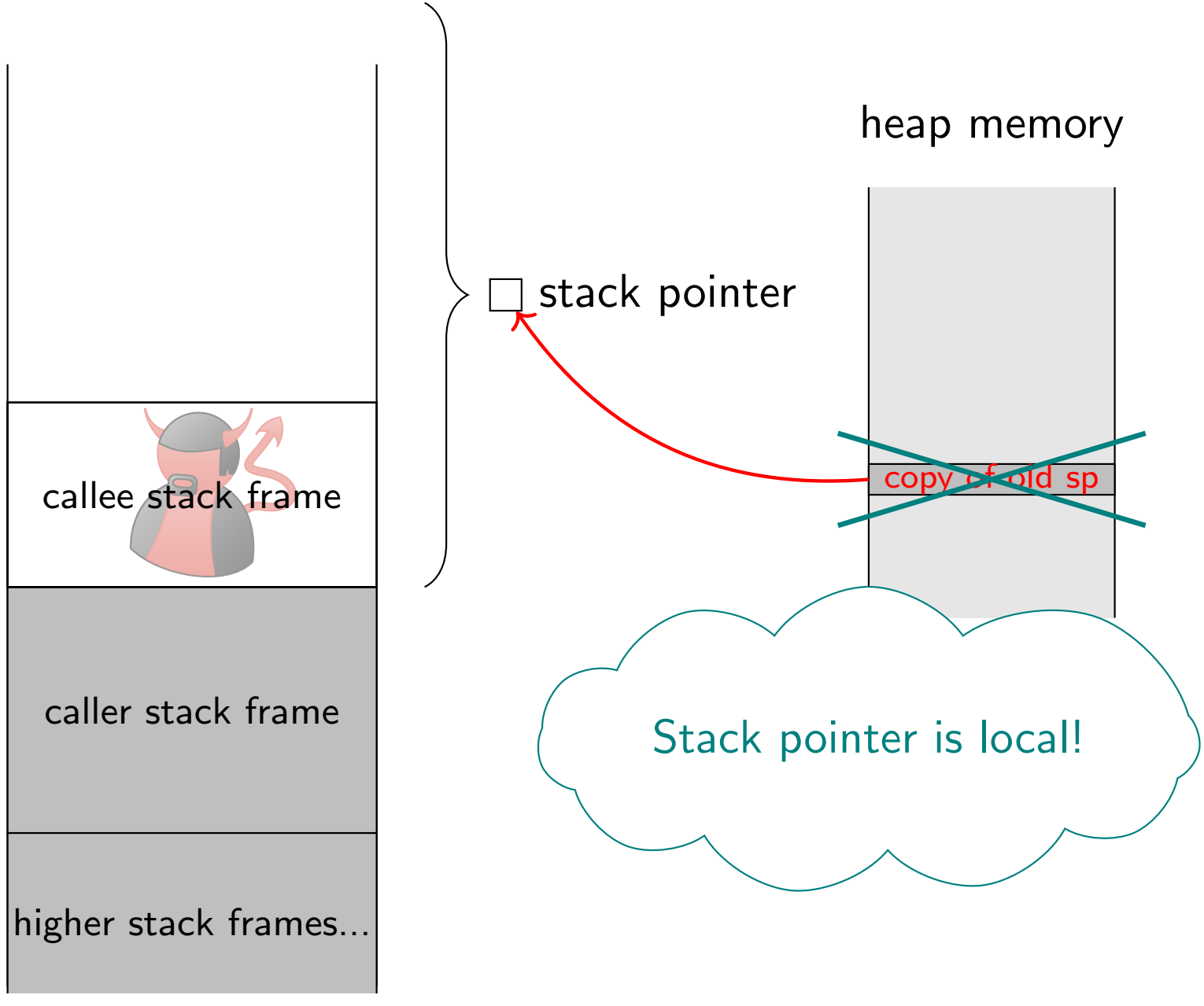
Stack and return capabilities: Attack 3



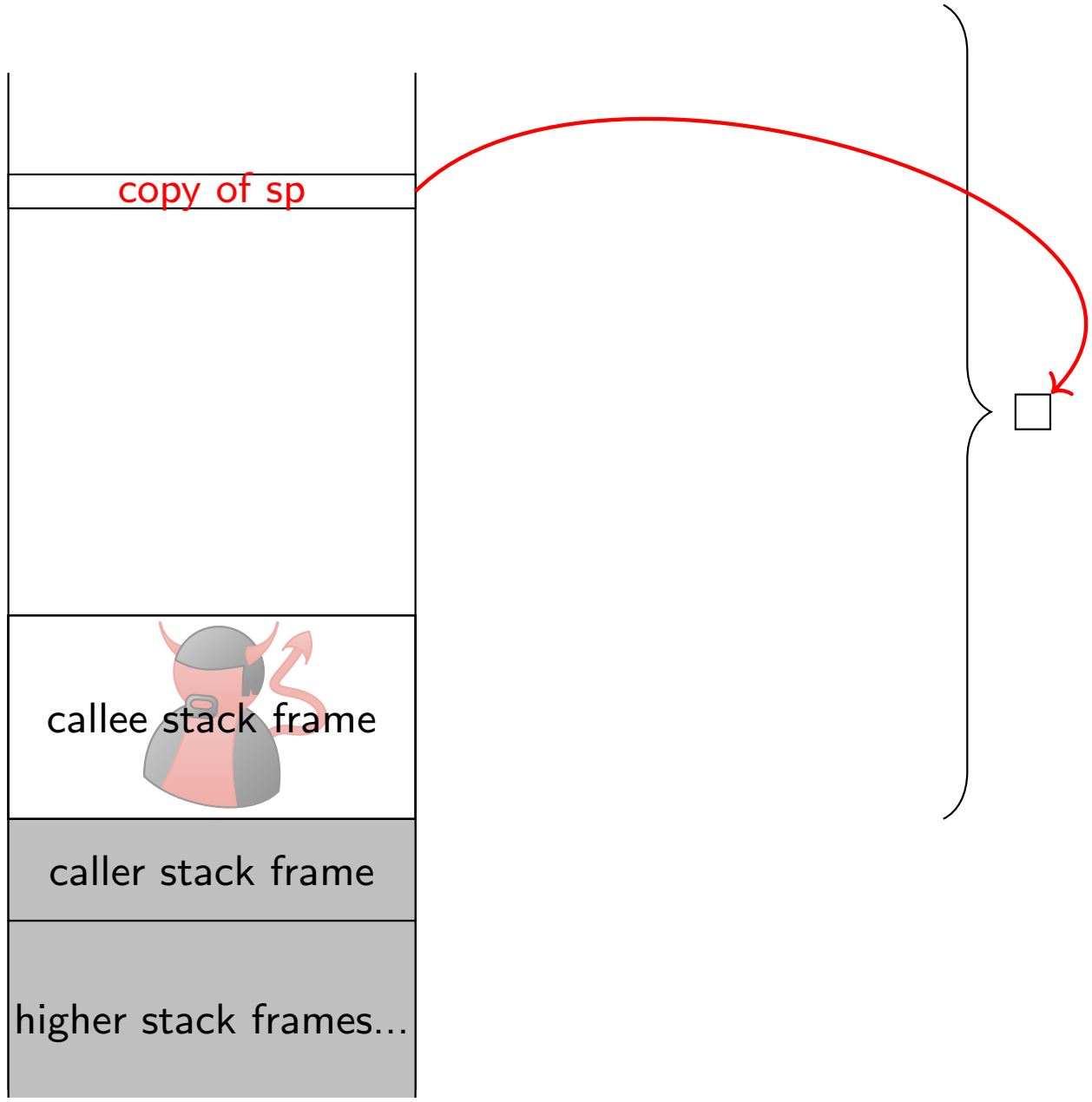
Stack and return pointer security

- Desirable properties:
  - Well-bracketed control flow
  - Stack frame encapsulation
- How to enforce?
  - Capabilities?
    - Not enough: three attacks!
  - CHERI Local capabilities?
    - Cannot leave registers

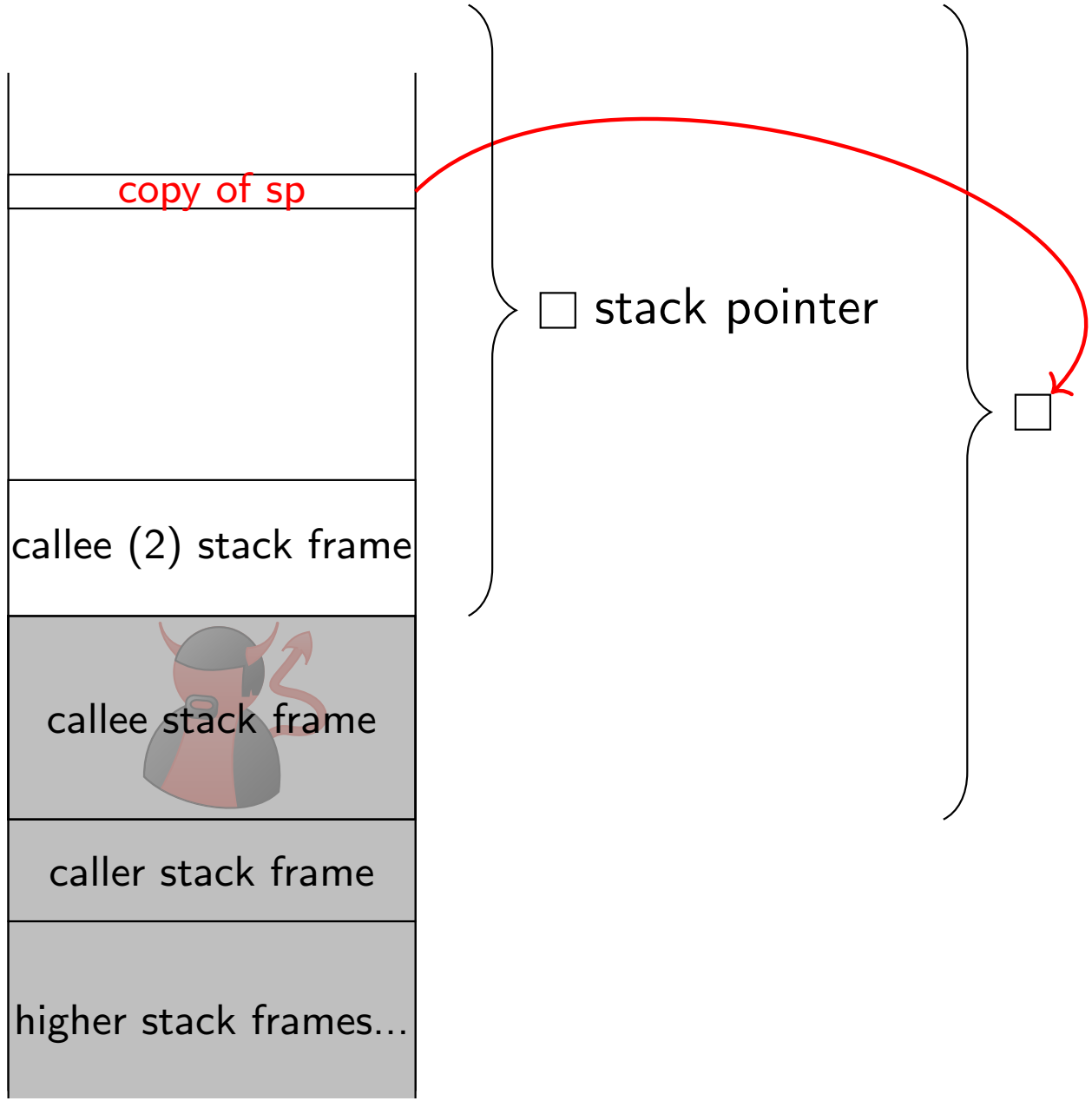
Local Stack Capabilities Prevent Attack 1



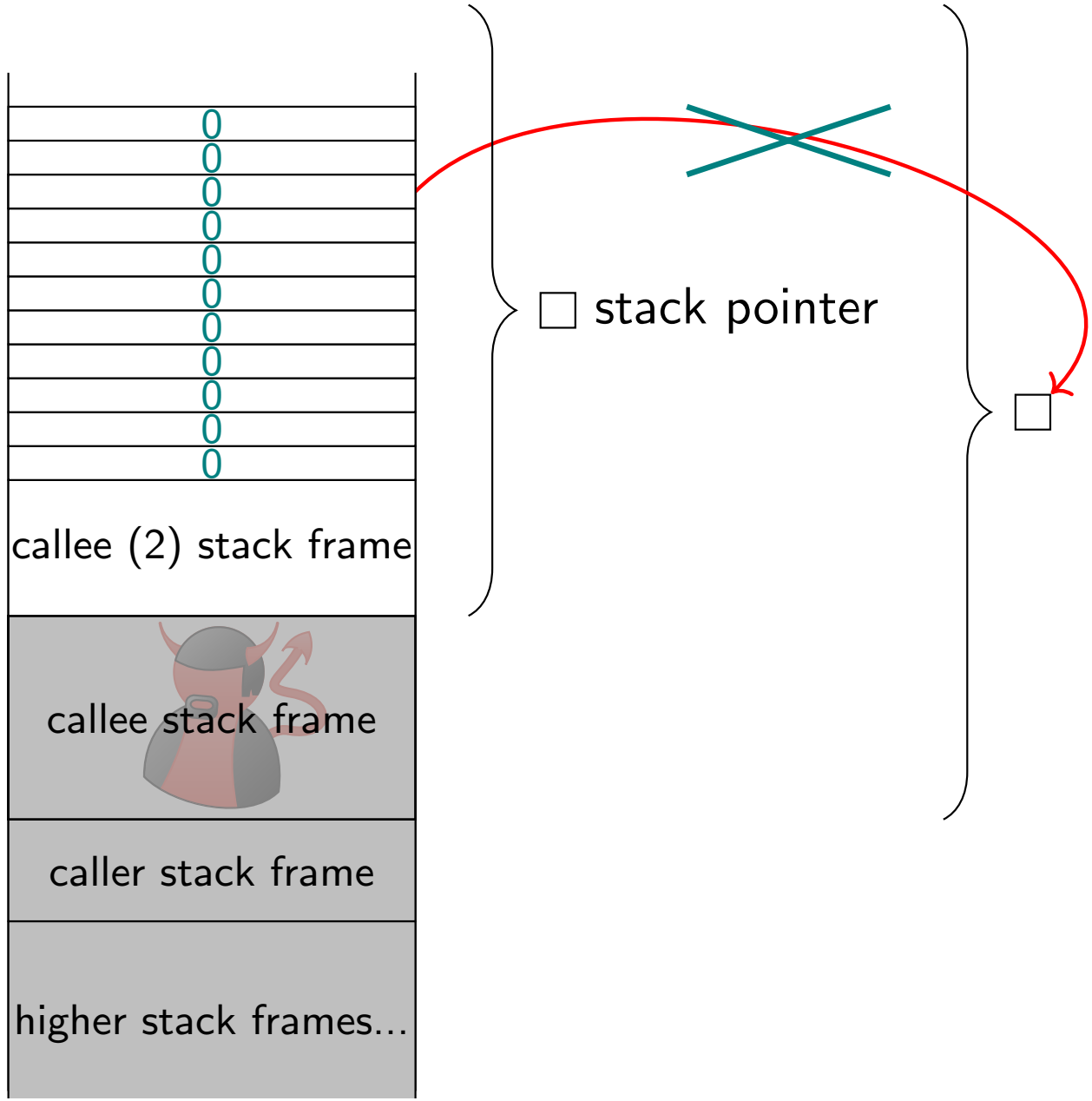
Local caps: Stack Clearing Prevents Attack 2



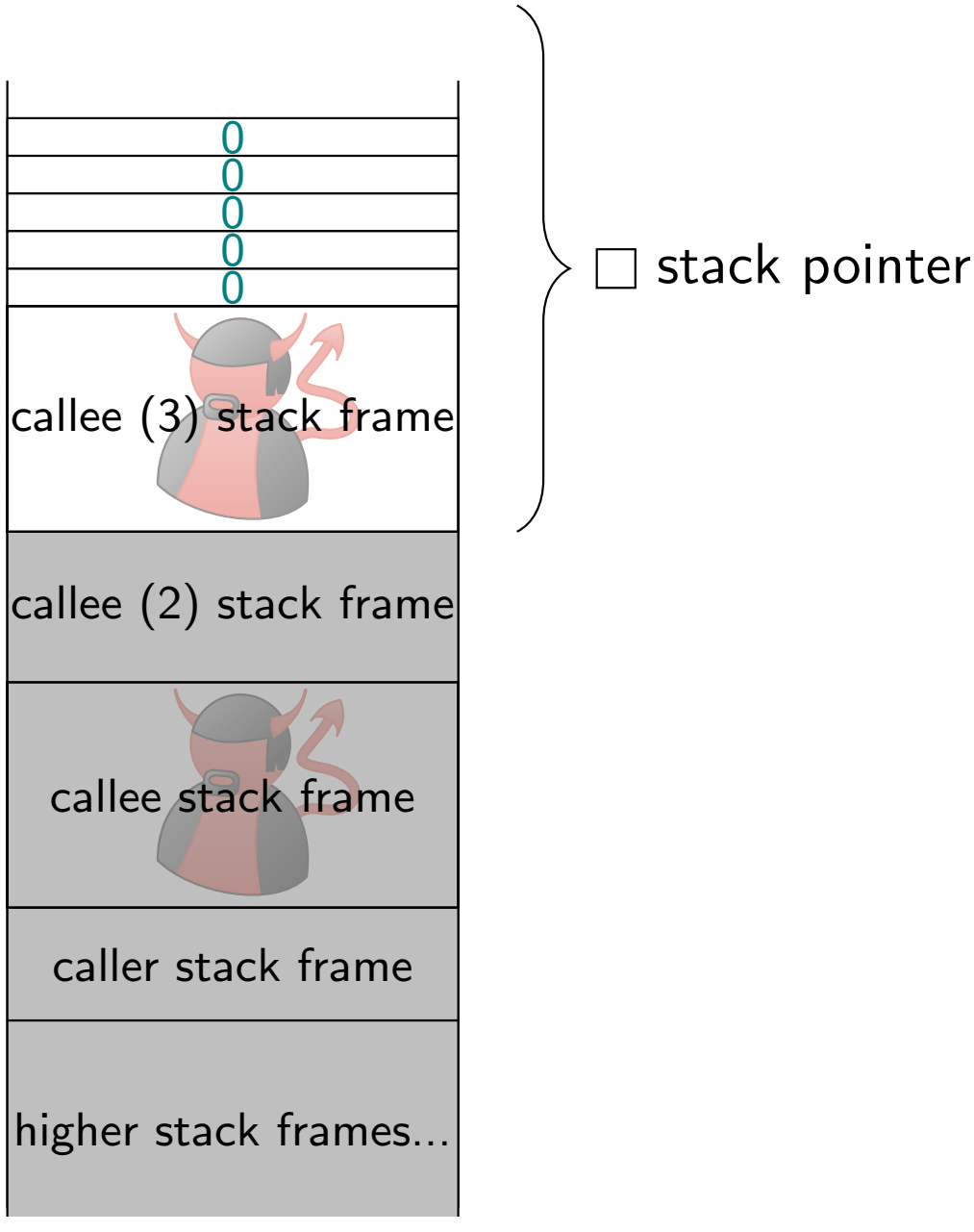
Local caps: Stack Clearing Prevents Attack 2



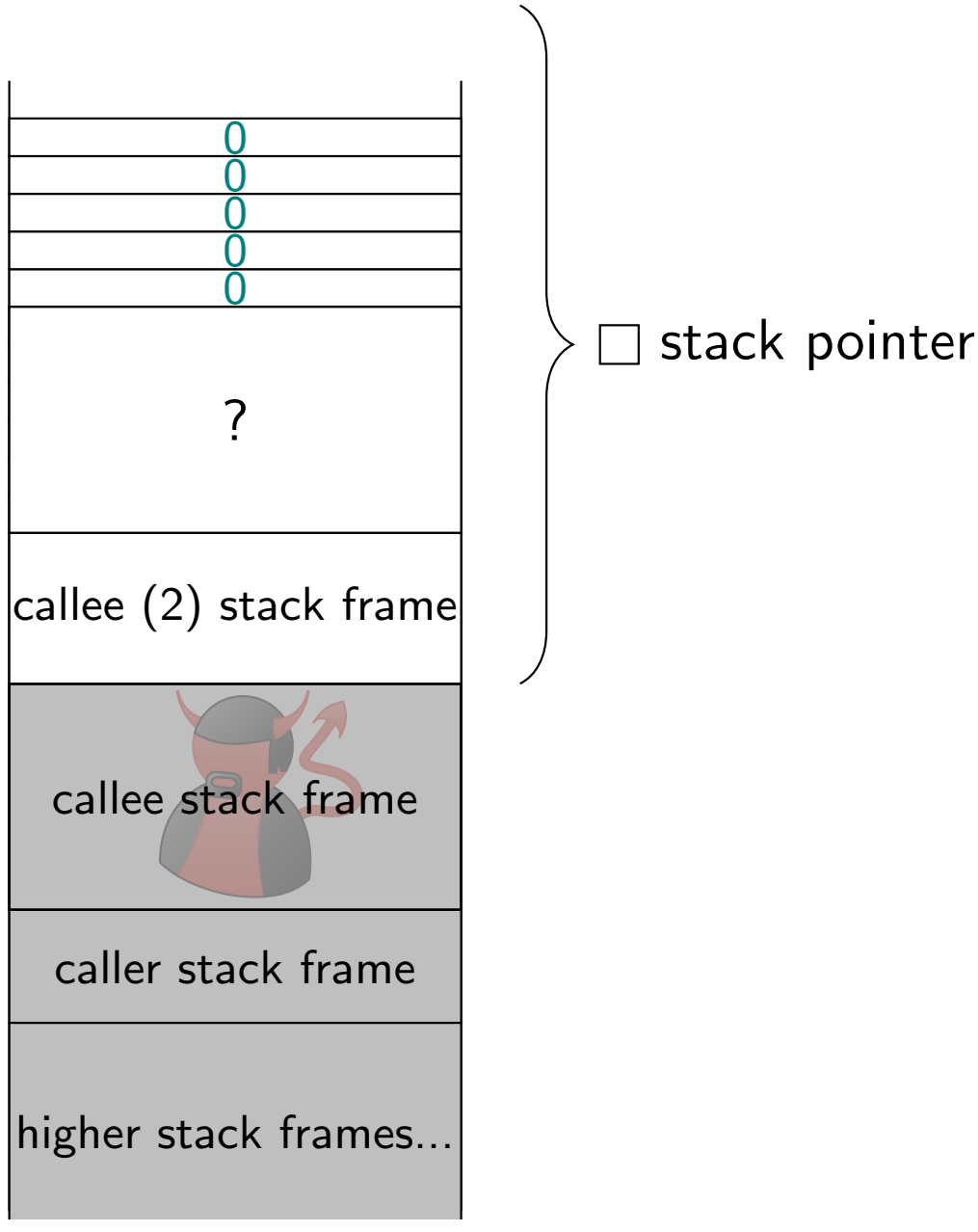
Local caps: Stack Clearing Prevents Attack 2



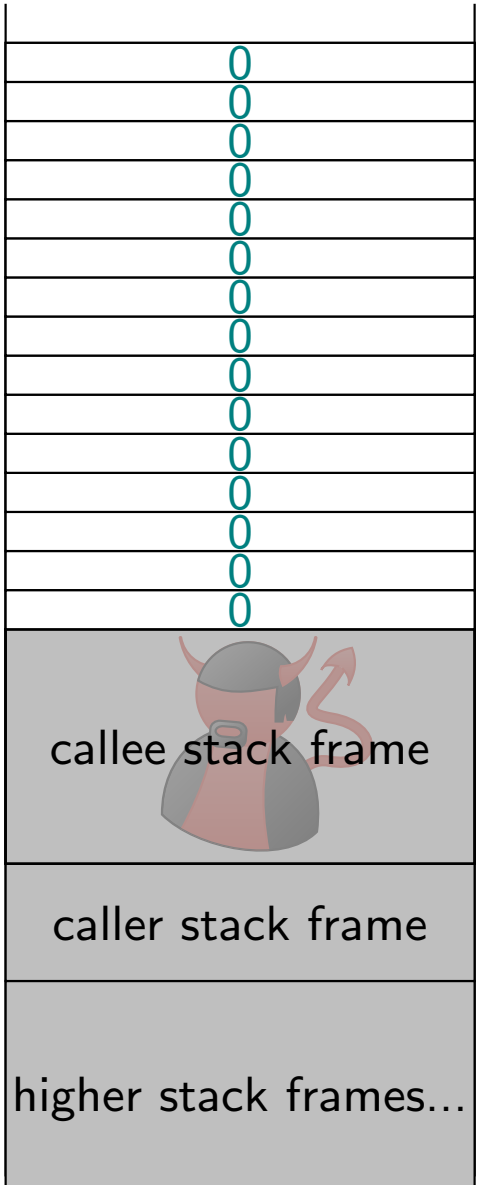
Local caps: Stack Clearing Prevents Attack 2



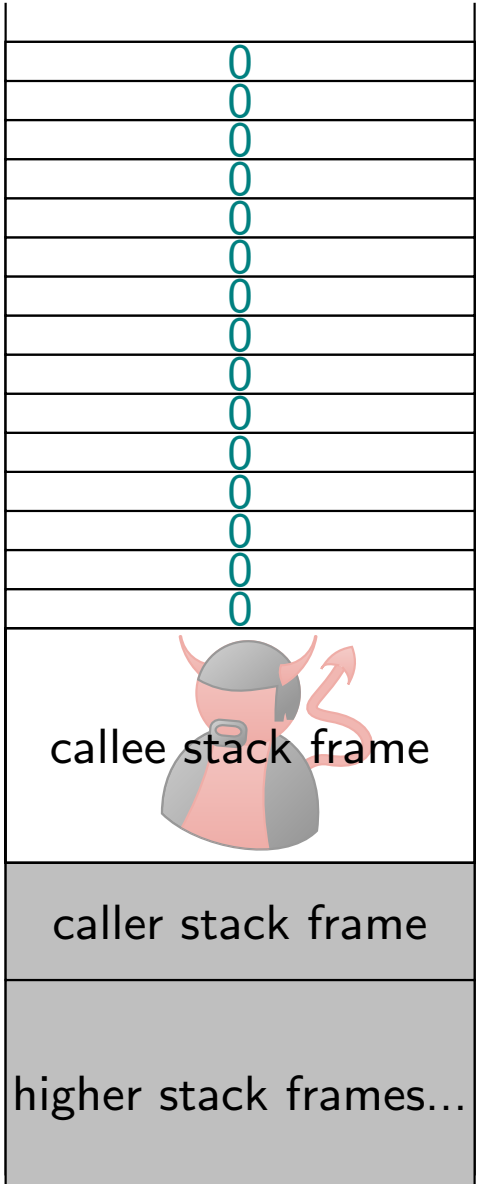
Local caps: Stack Clearing Prevents Attack 2



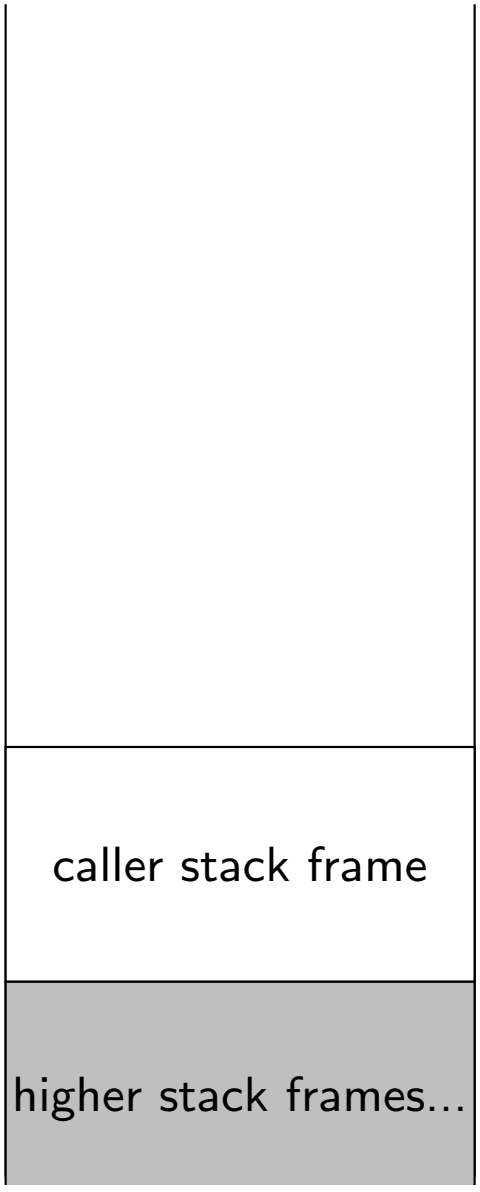
Local caps: Stack Clearing Prevents Attack 2

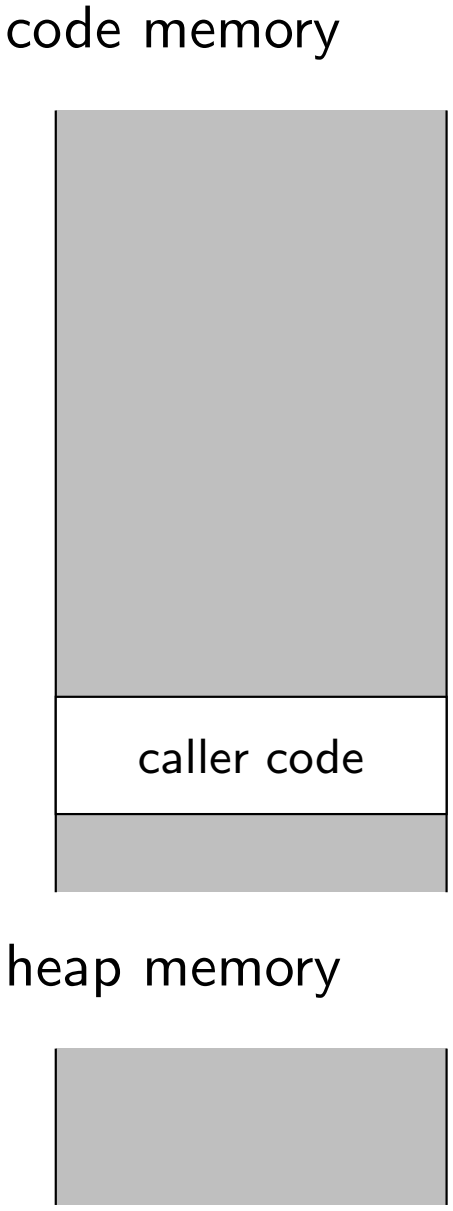


Local caps: Stack Clearing Prevents Attack 2

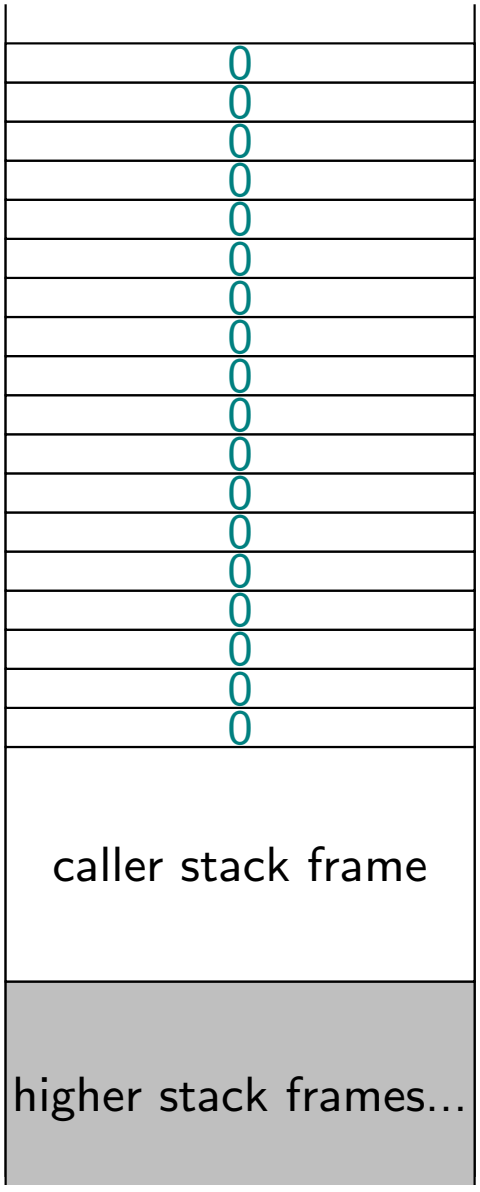


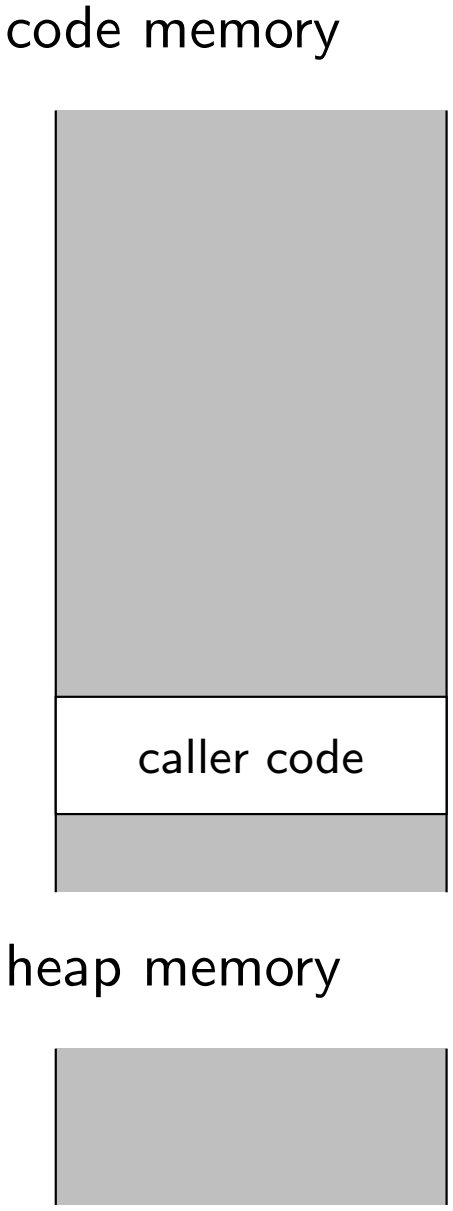
Local Stack Capabilities Prevent Attack 3



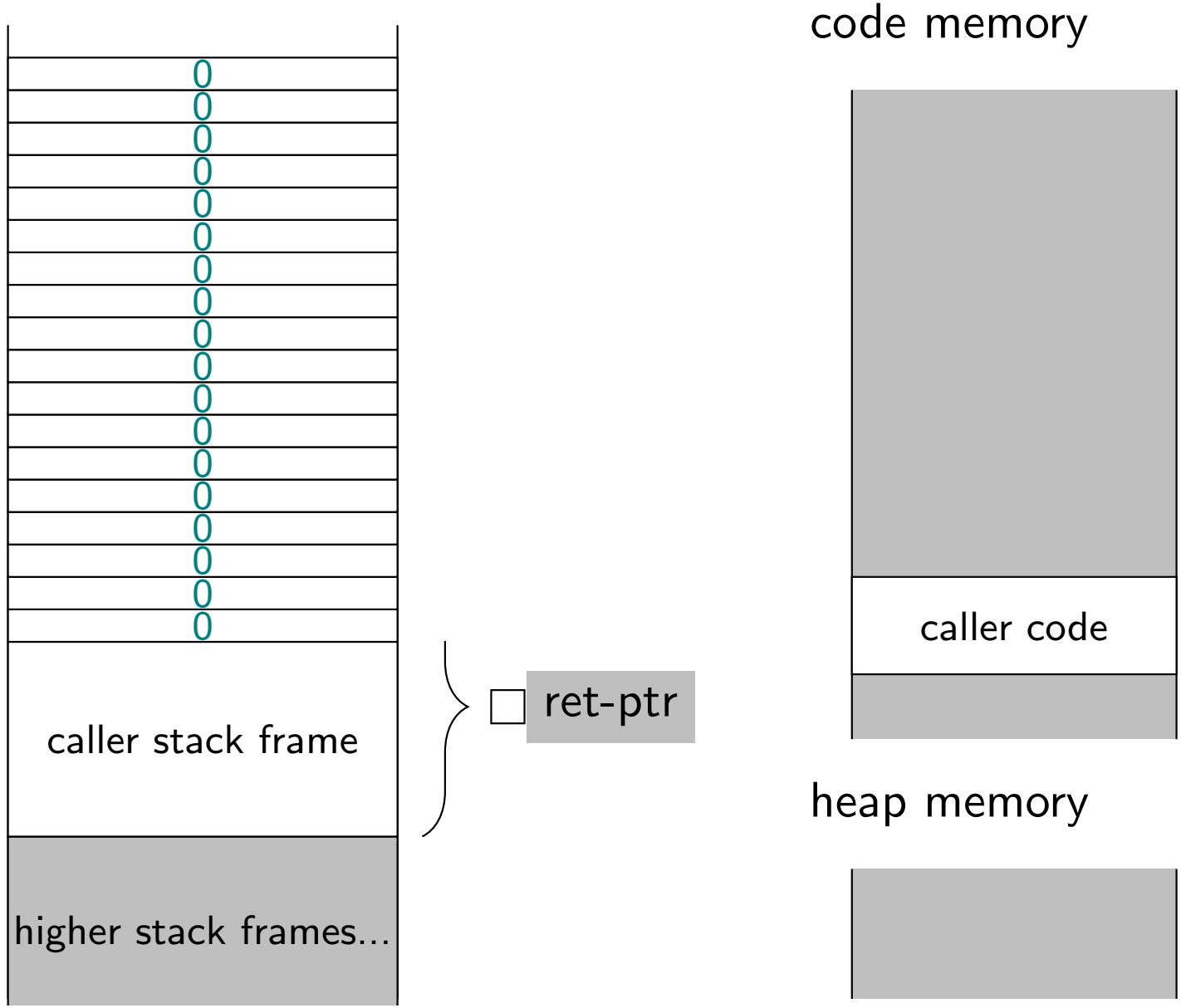


Local Stack Capabilities Prevent Attack 3

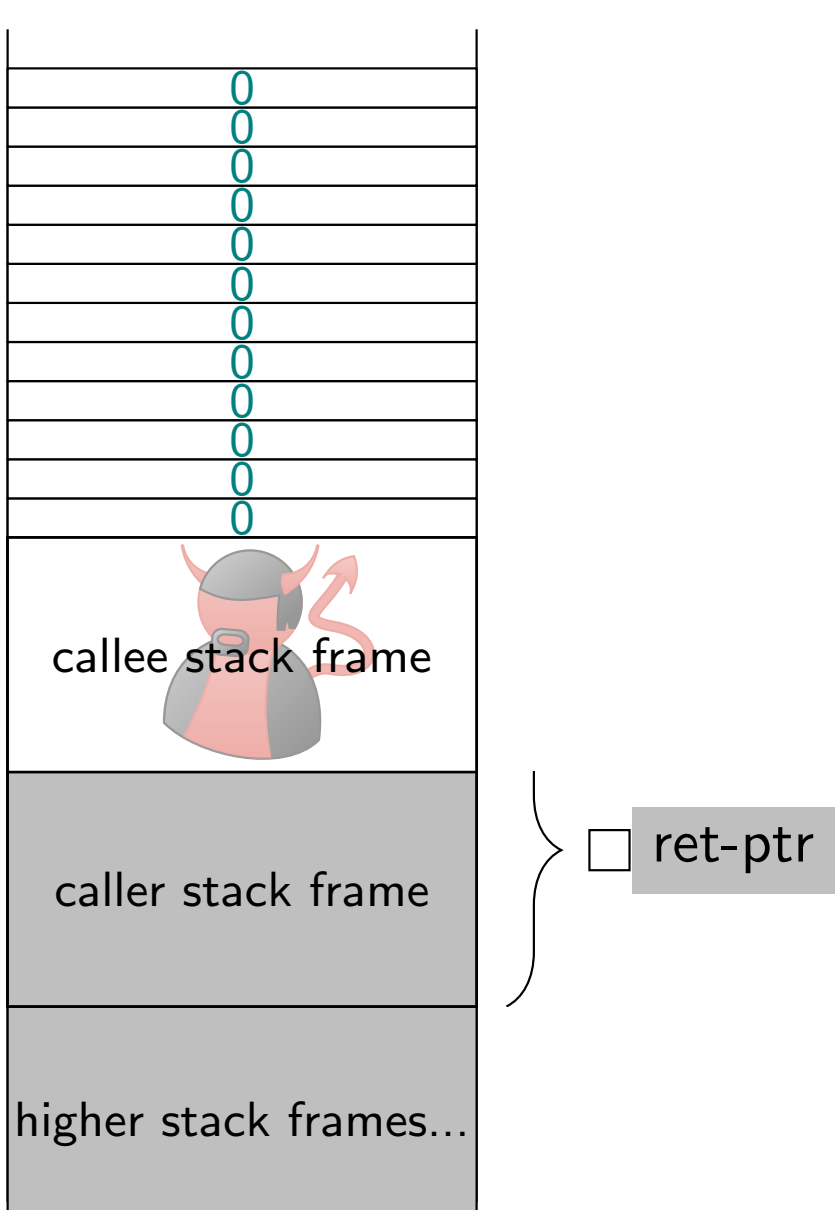




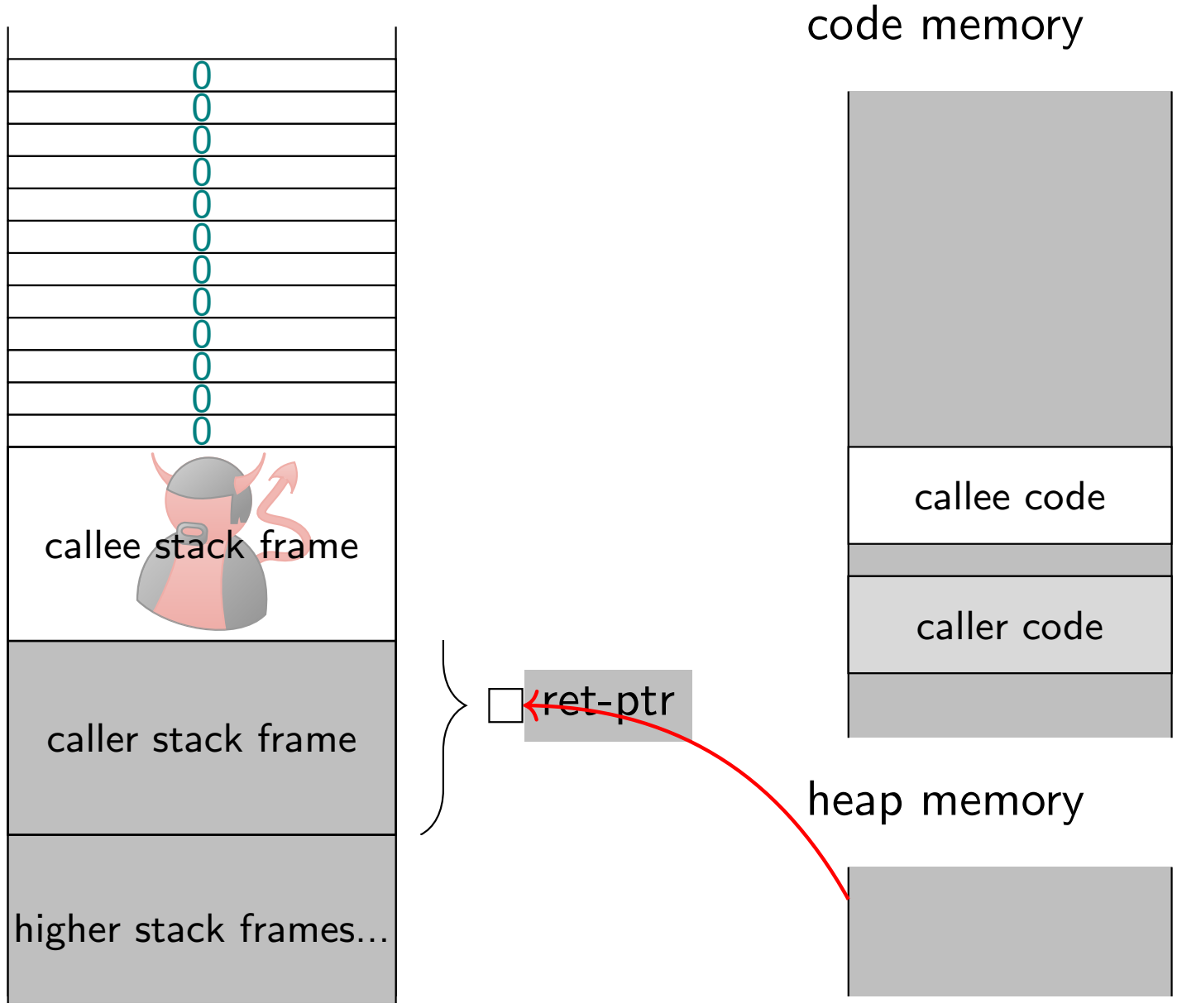
Local Stack Capabilities Prevent Attack 3



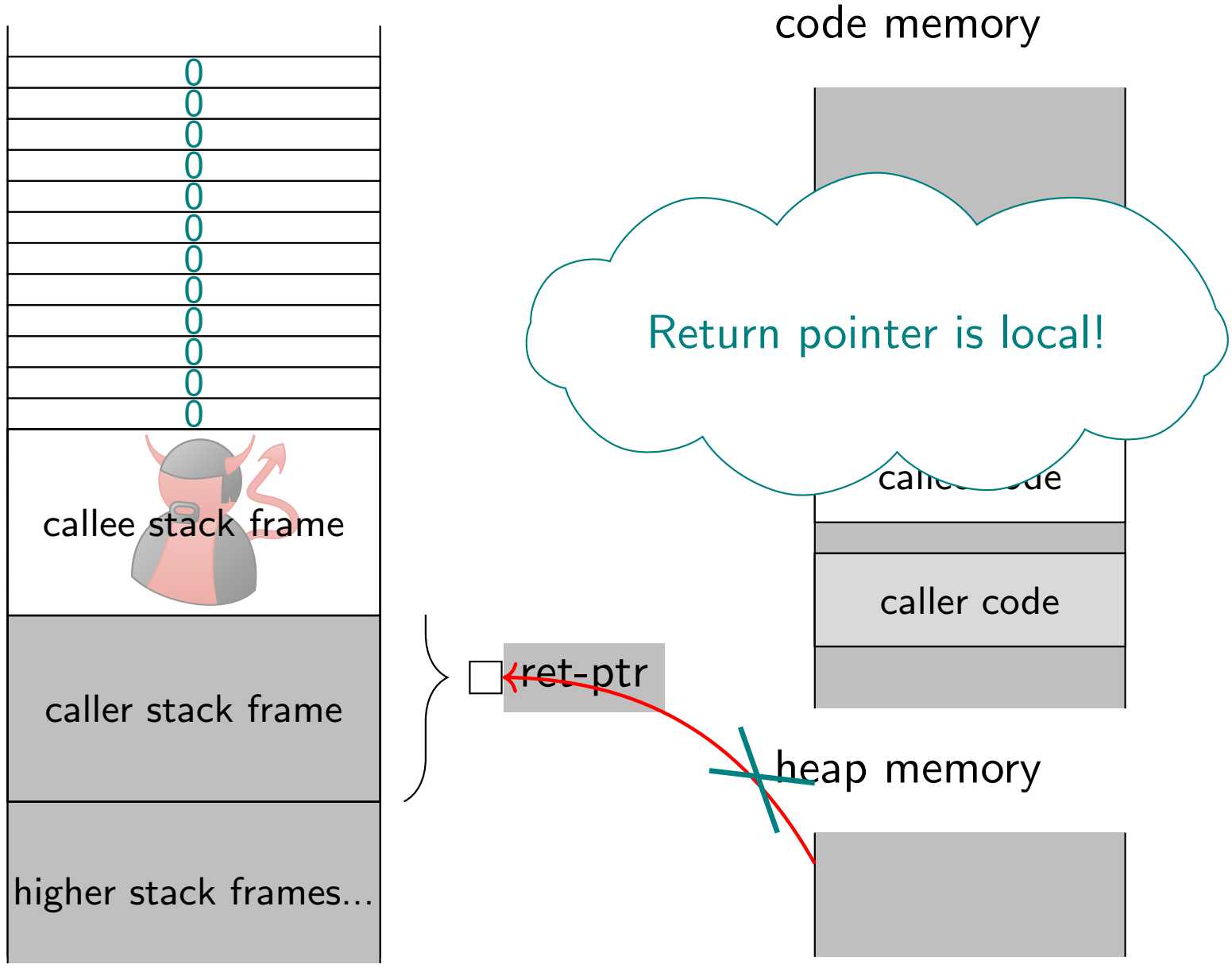
Local Stack Capabilities Prevent Attack 3



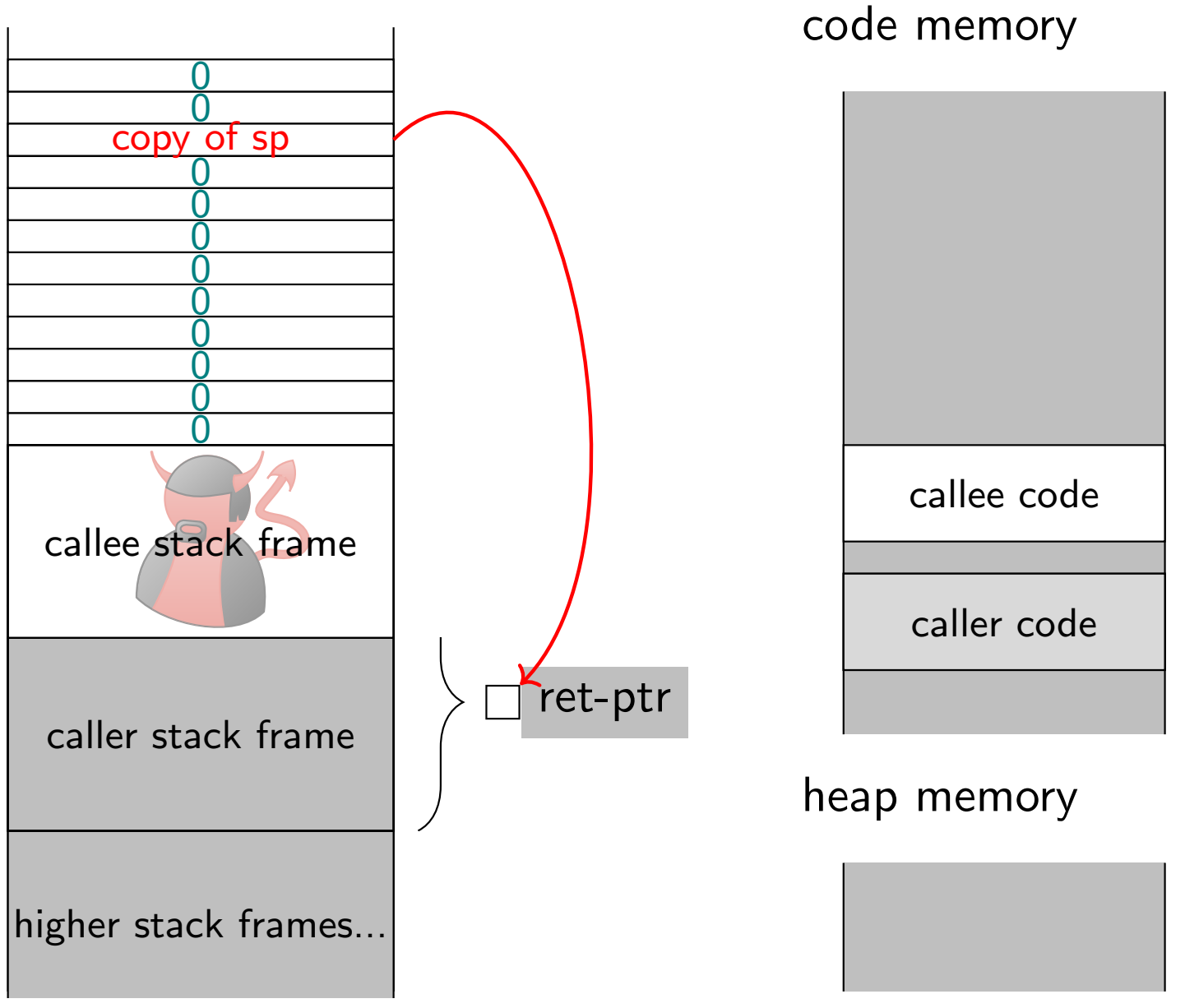
Local Stack Capabilities Prevent Attack 3



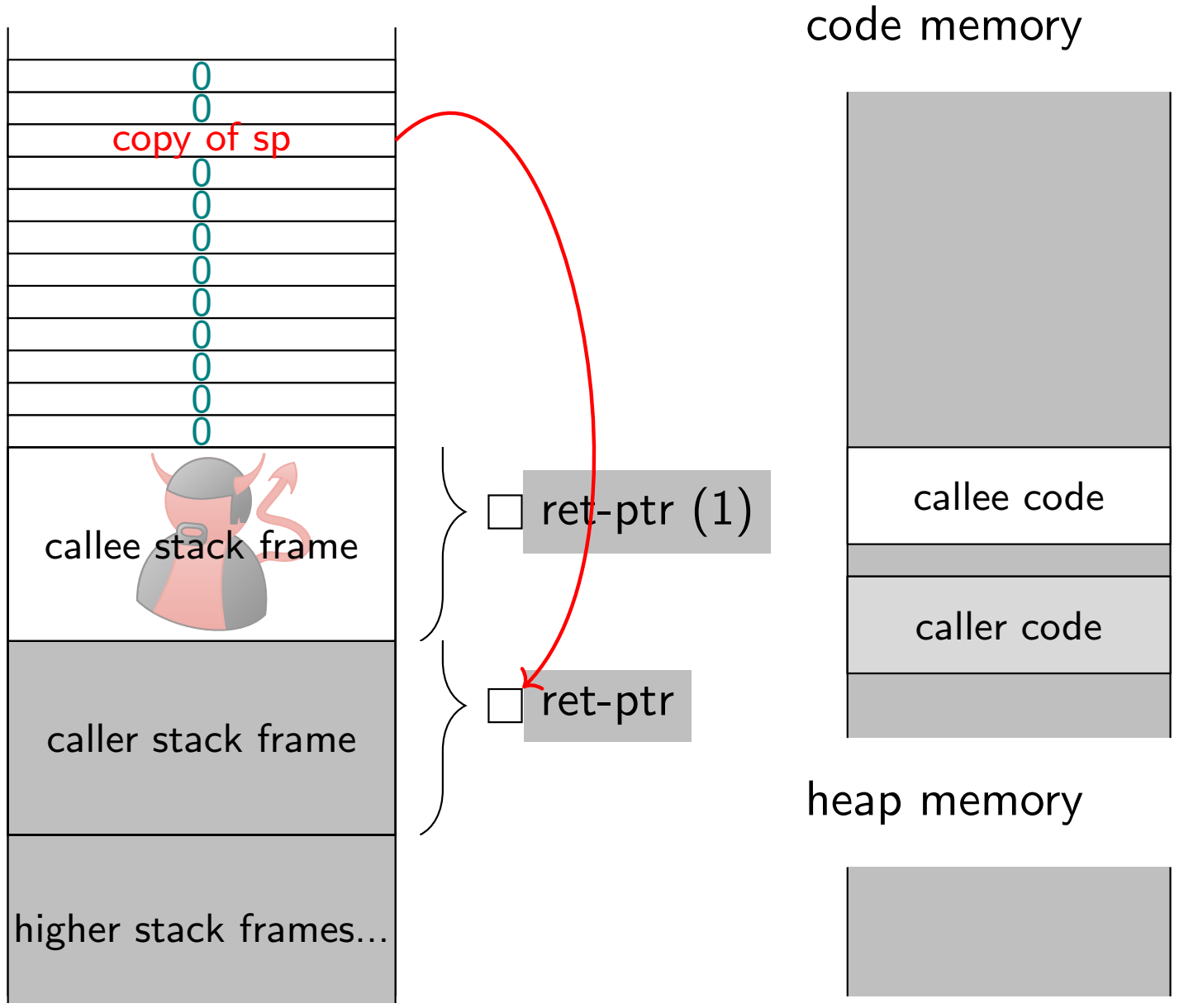
Local Stack Capabilities Prevent Attack 3



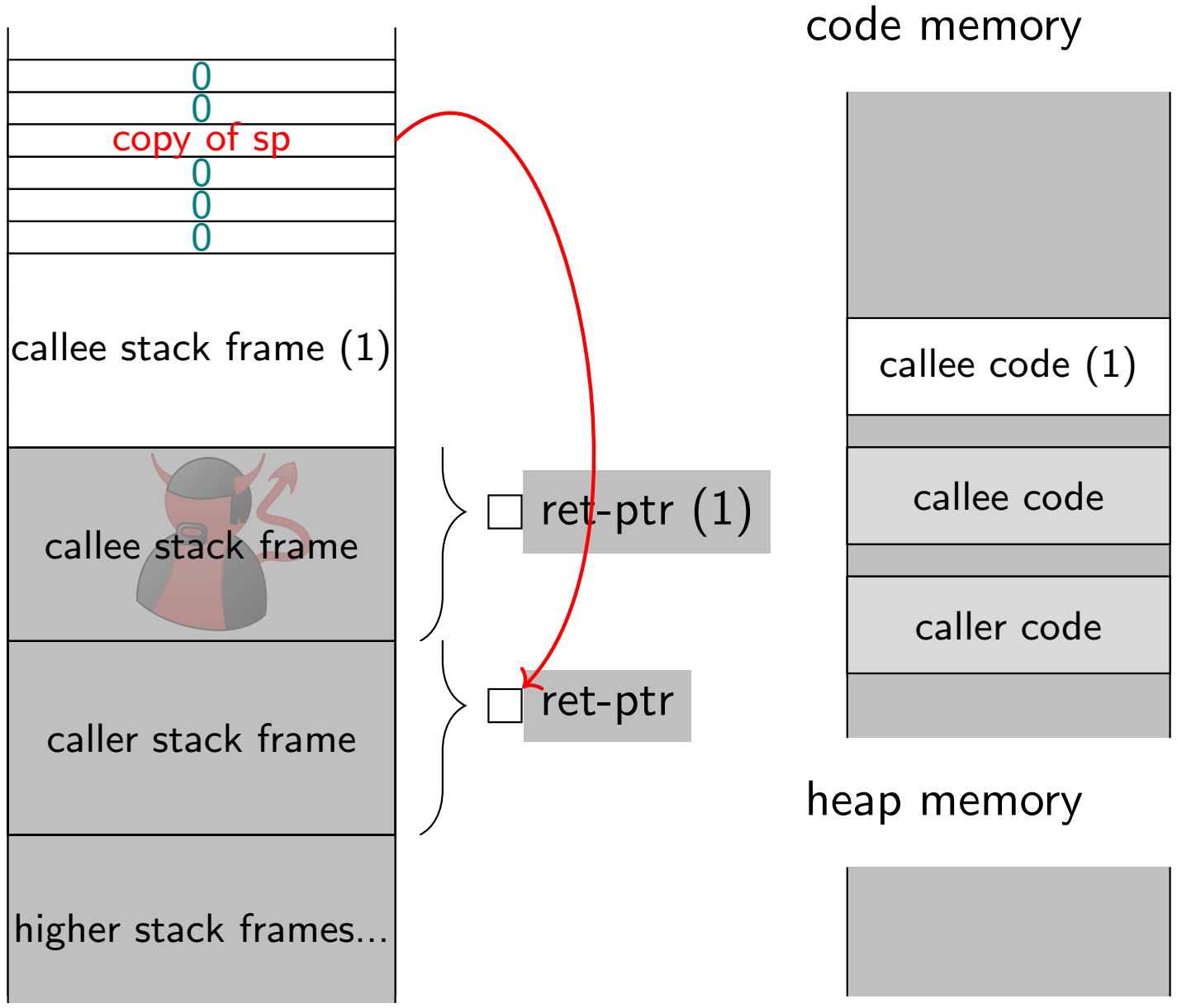
Local Stack Capabilities Prevent Attack 3



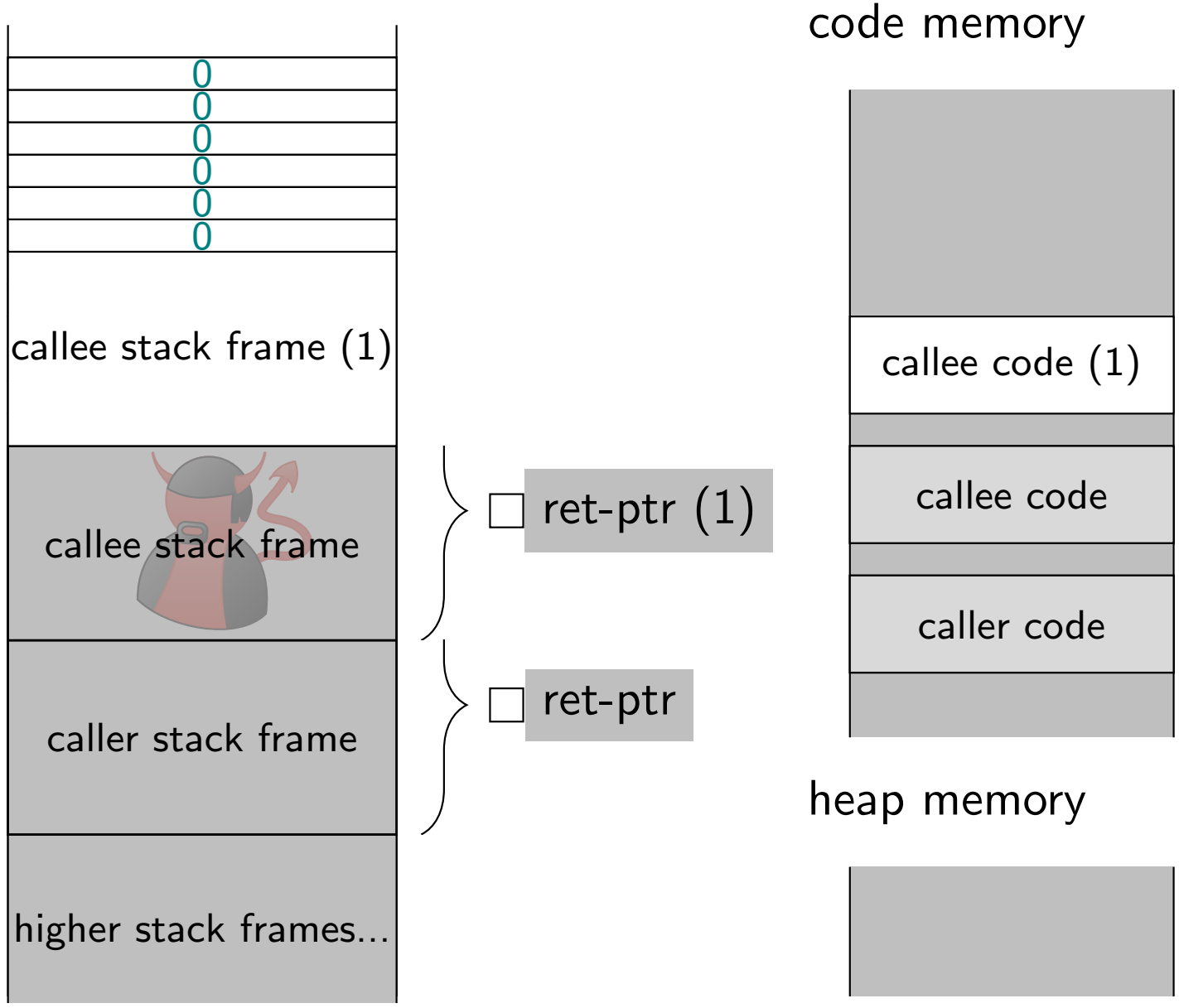
Local Stack Capabilities Prevent Attack 3



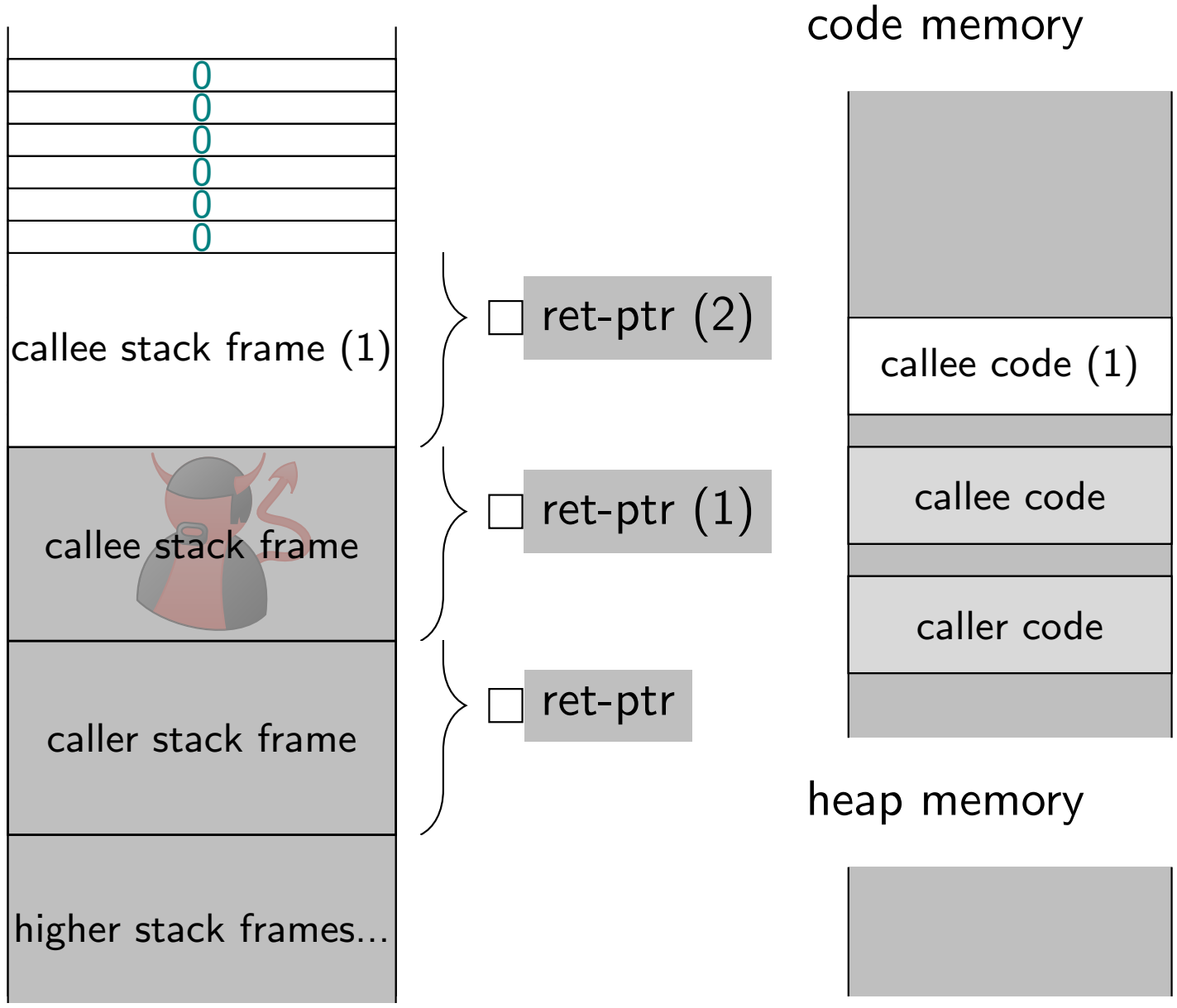
Local Stack Capabilities Prevent Attack 3



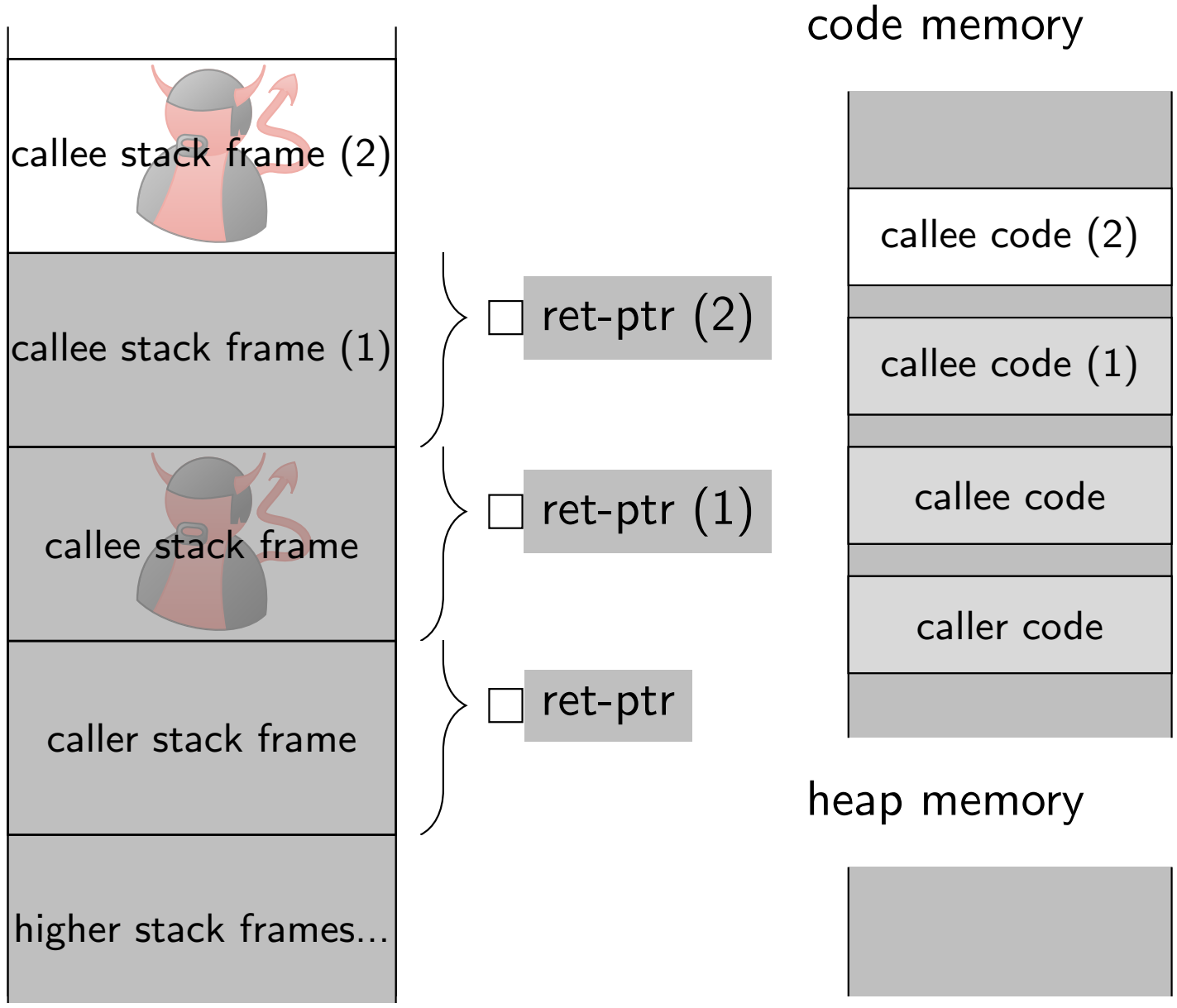
Local Stack Capabilities Prevent Attack 3



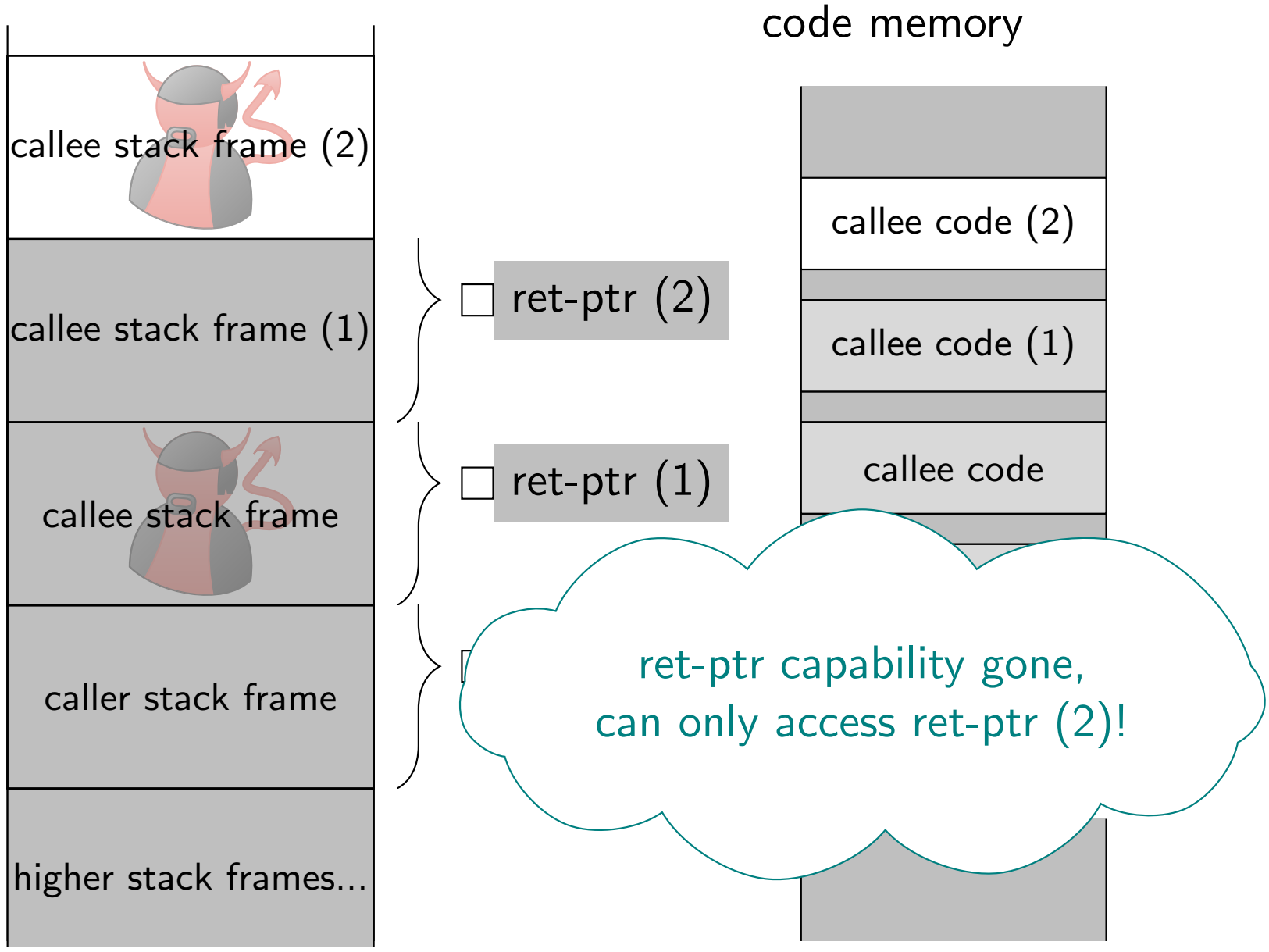
Local Stack Capabilities Prevent Attack 3



Local Stack Capabilities Prevent Attack 3



Local Stack Capabilities Prevent Attack 3



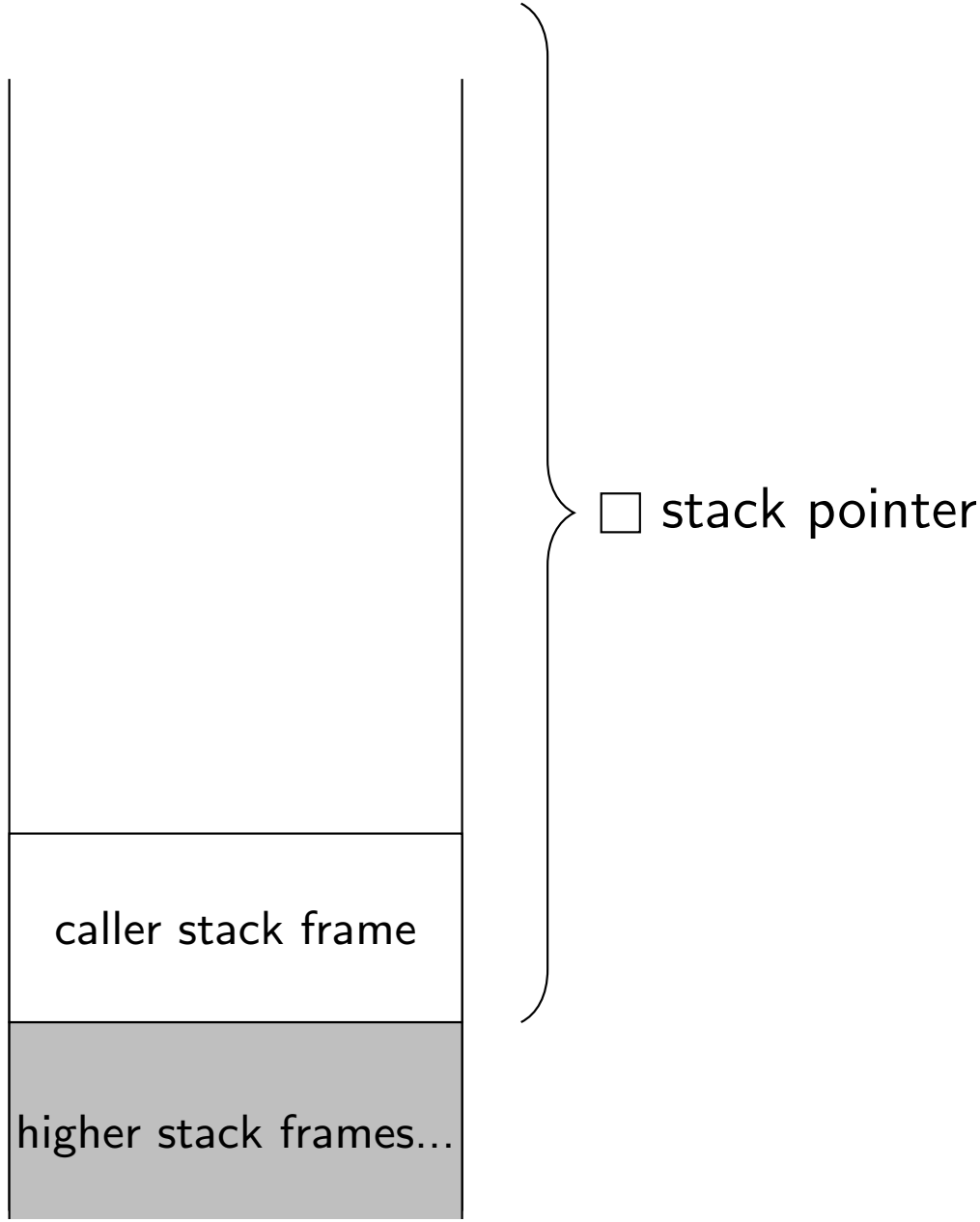
Stack and return pointer security

- Desirable properties:
  - Well-bracketed control flow
  - Stack frame encapsulation
- How to enforce?
  - Capabilities?
    - Not enough: three attacks!
  - CHERI Local capabilities?
    - Cannot leave registers
    - Works, but... [Skorstengaard et al., ESOP 2018]

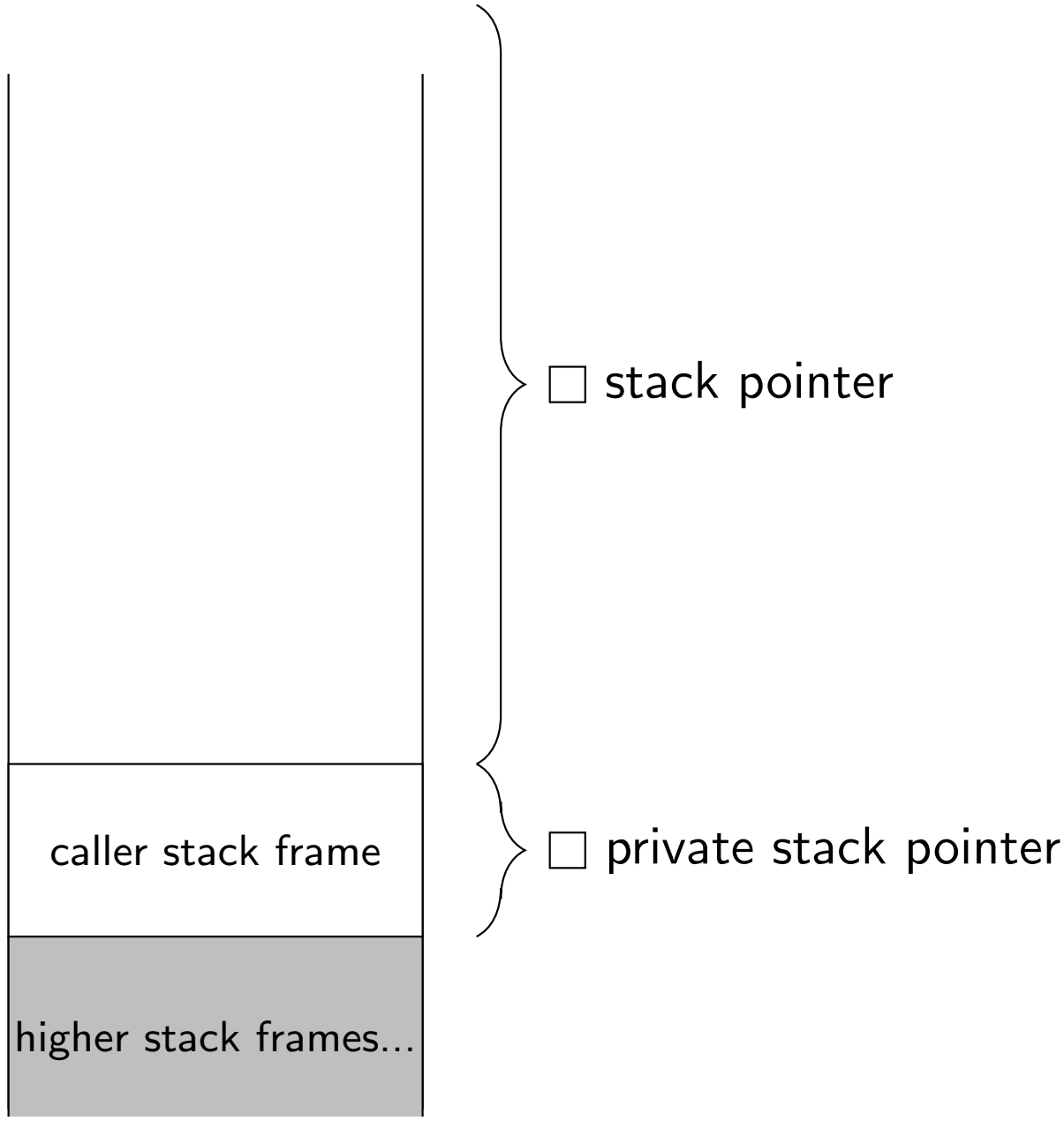
Stack and return pointer security

- Desirable properties:
  - Well-bracketed control flow
  - Stack frame encapsulation
- How to enforce?
  - Capabilities?
    - Not enough: three attacks!
  - CHERI Local capabilities?
    - Cannot leave registers
    - Works, but... [Skorstengaard et al., ESOP 2018]
  - CHERI Linear capabilities?
    - Non-copyable capabilities

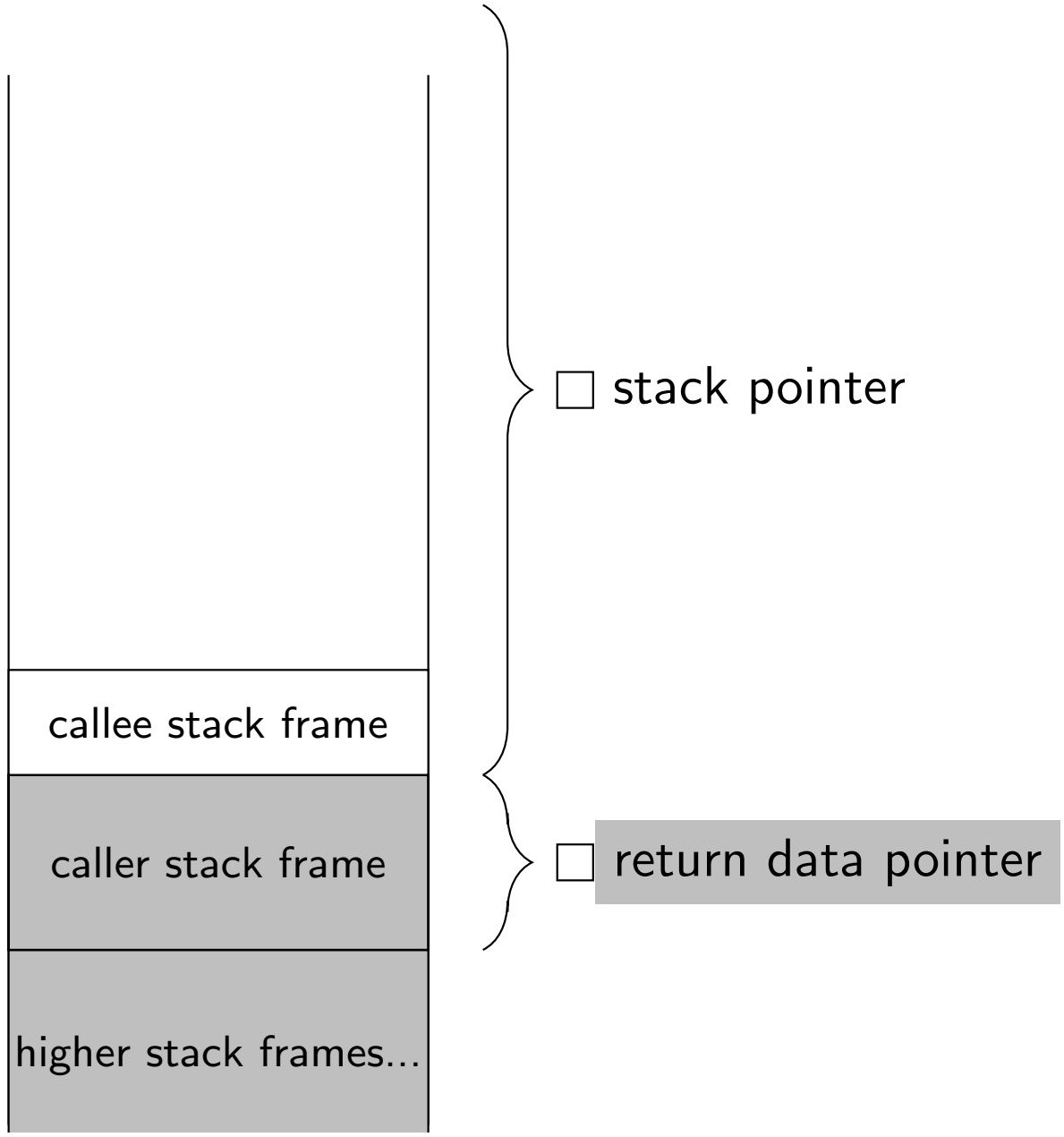
Linear Stack Capabilities



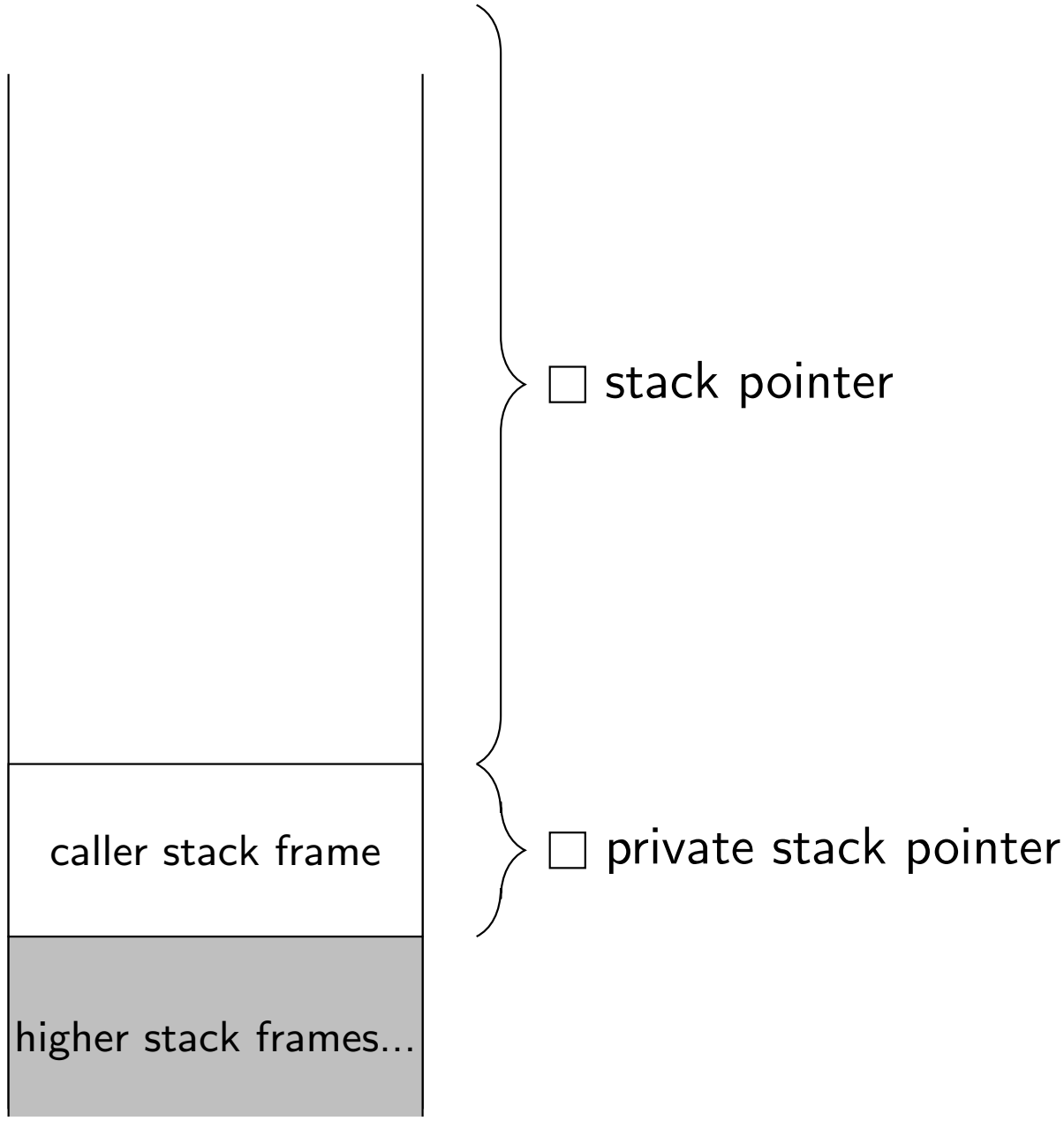
Linear Stack Capabilities



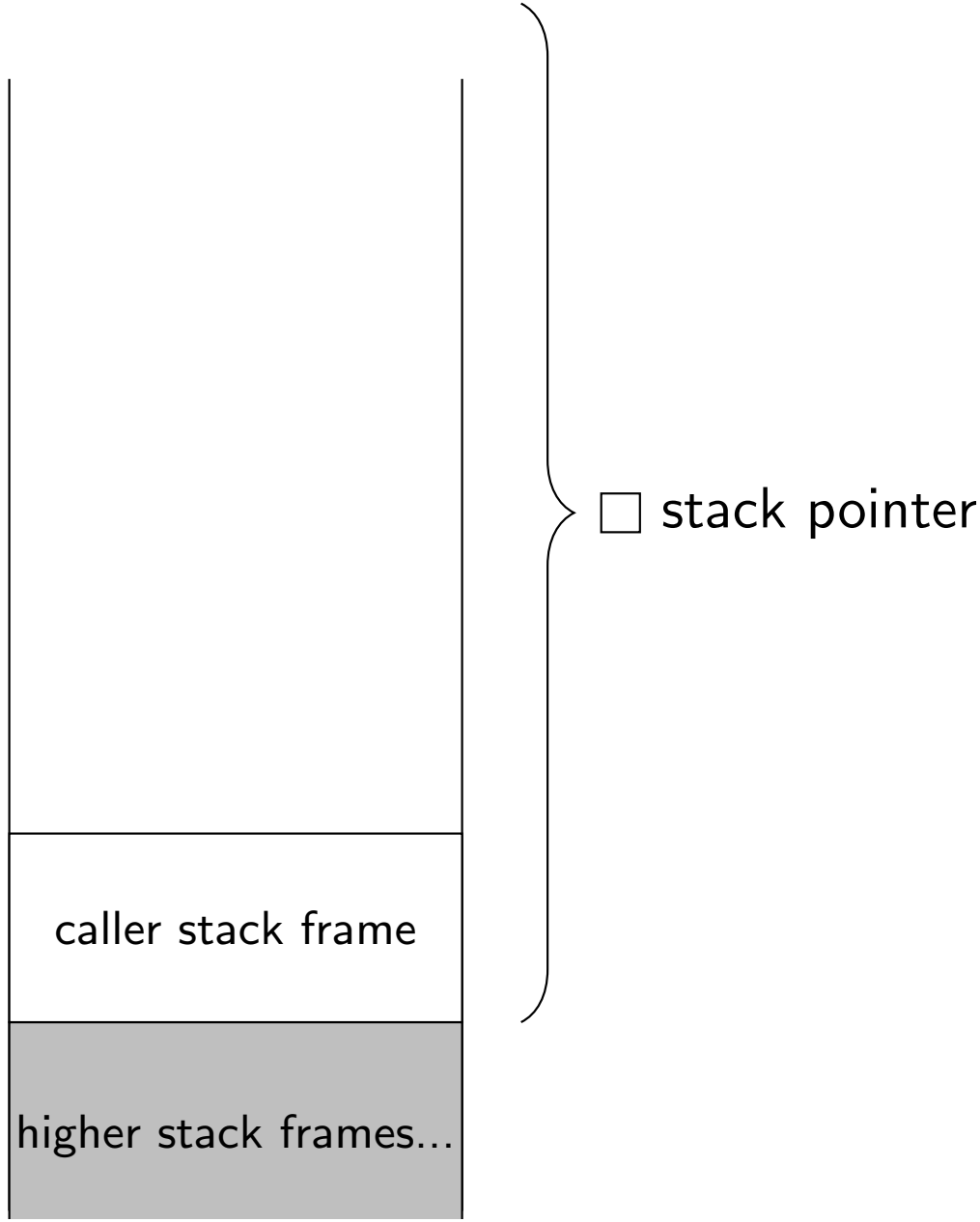
Linear Stack Capabilities



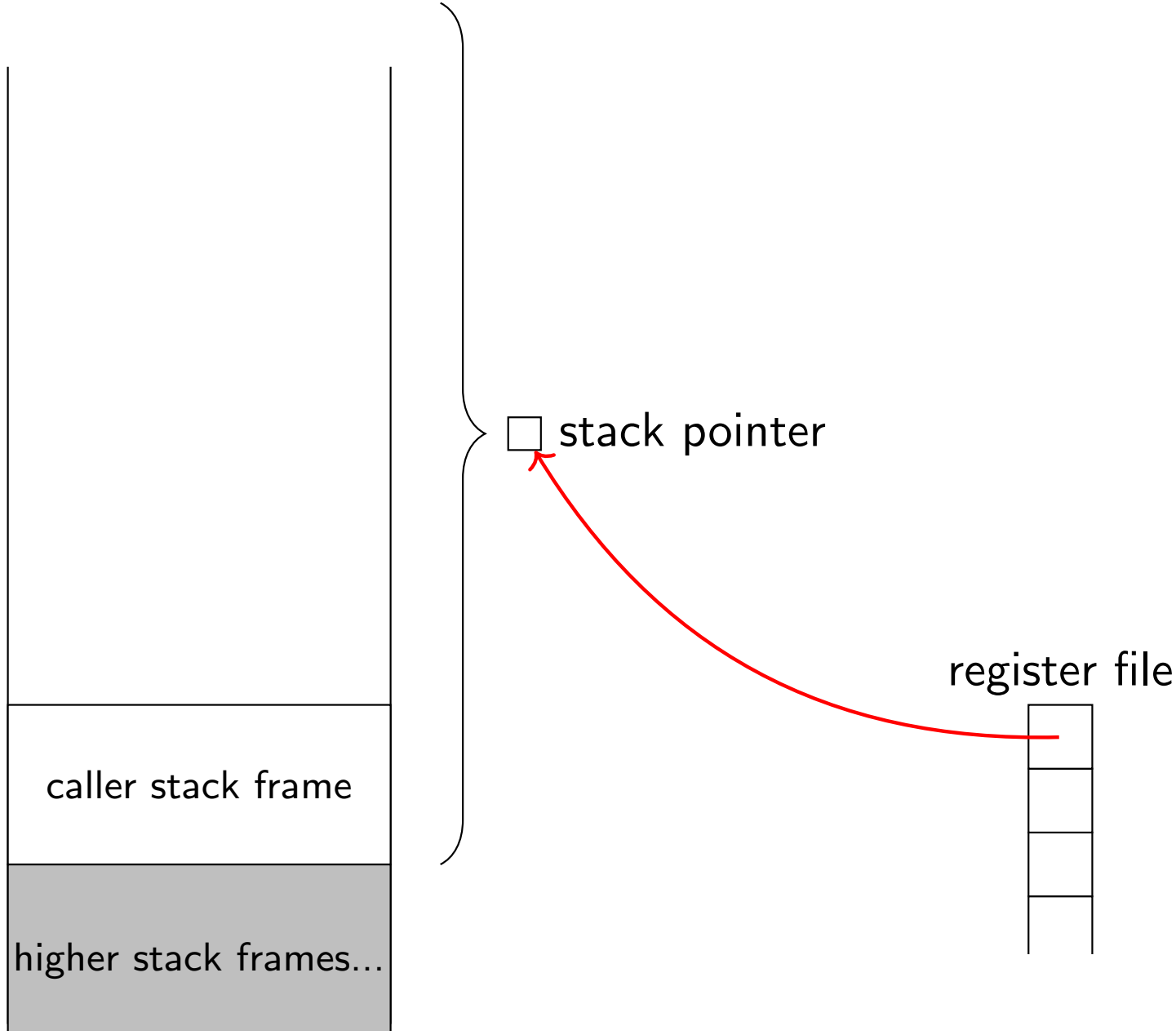
Linear Stack Capabilities



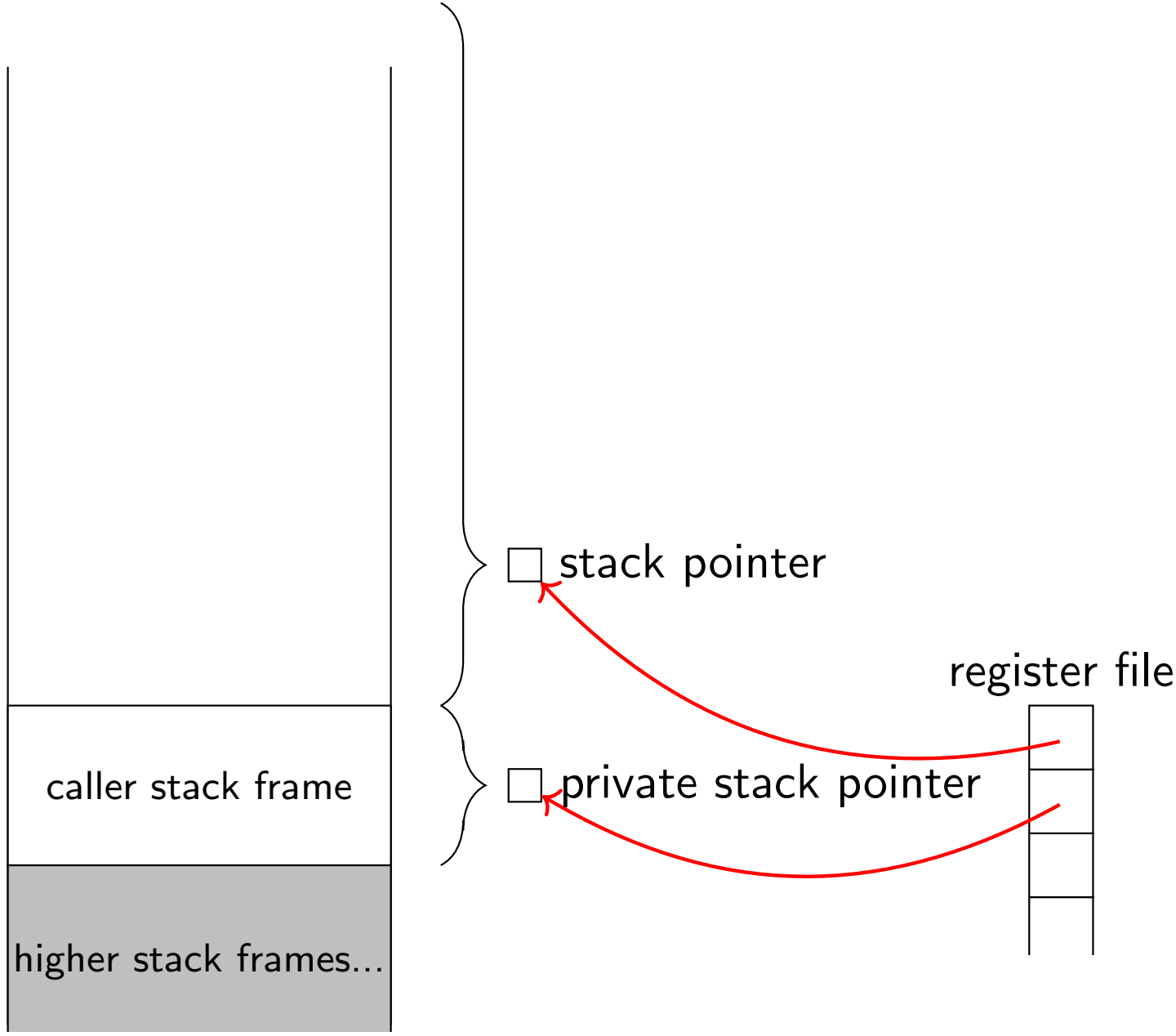
Linear Stack Capabilities



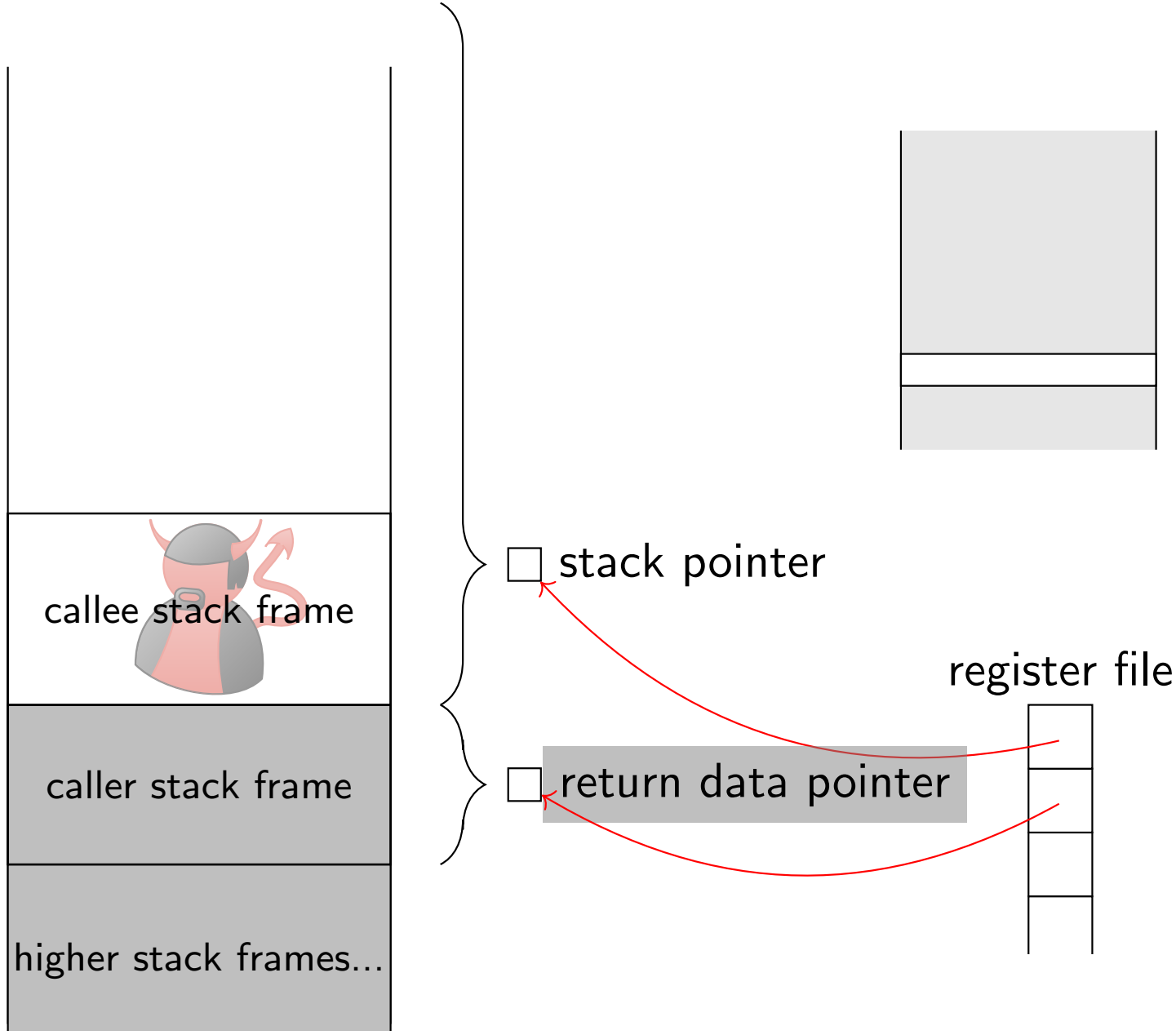
Linear Capabilities Prevent Attack 1



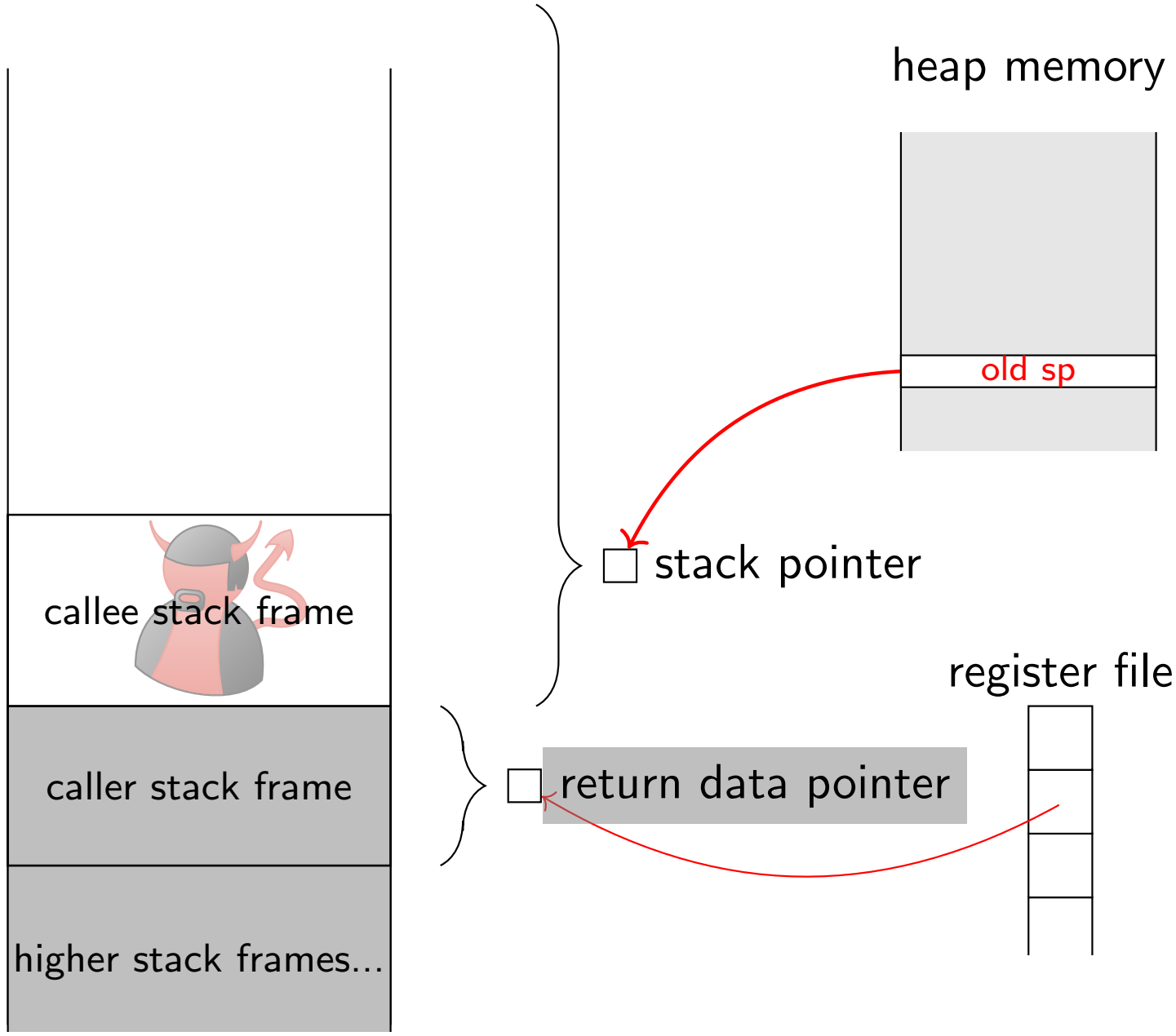
Linear Capabilities Prevent Attack 1



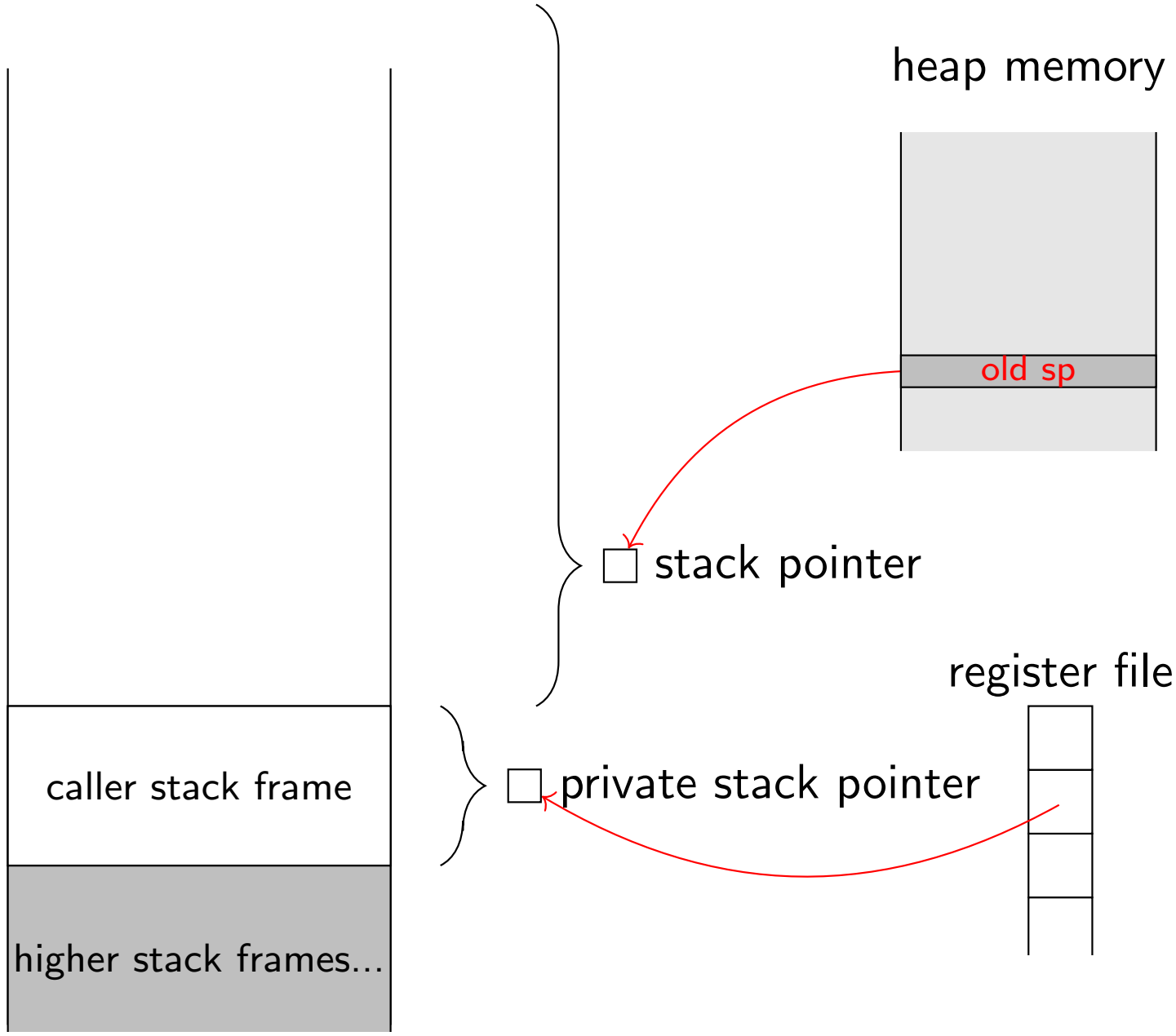
Linear Capabilities Prevent Attack 1



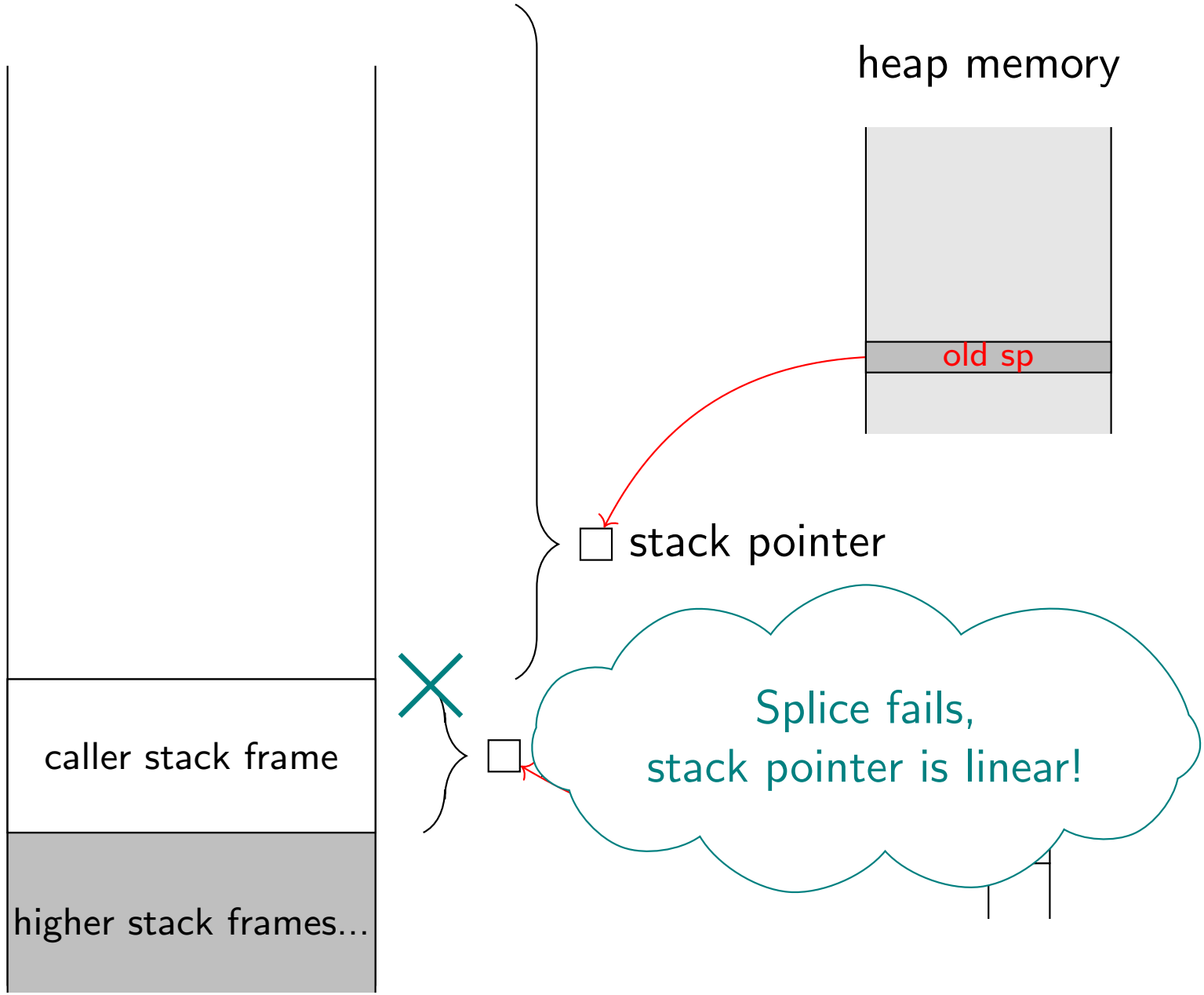
Linear Capabilities Prevent Attack 1



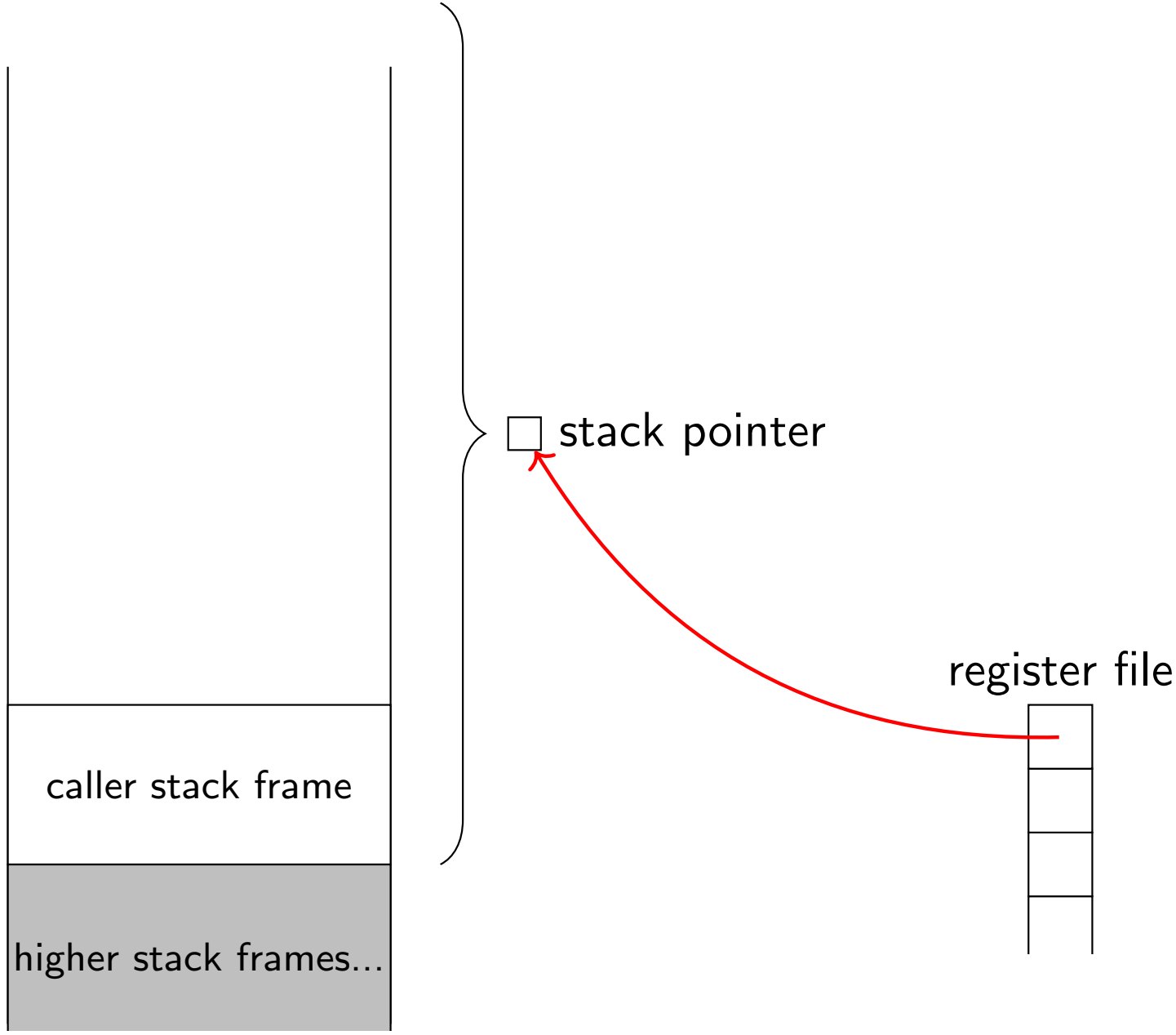
Linear Capabilities Prevent Attack 1



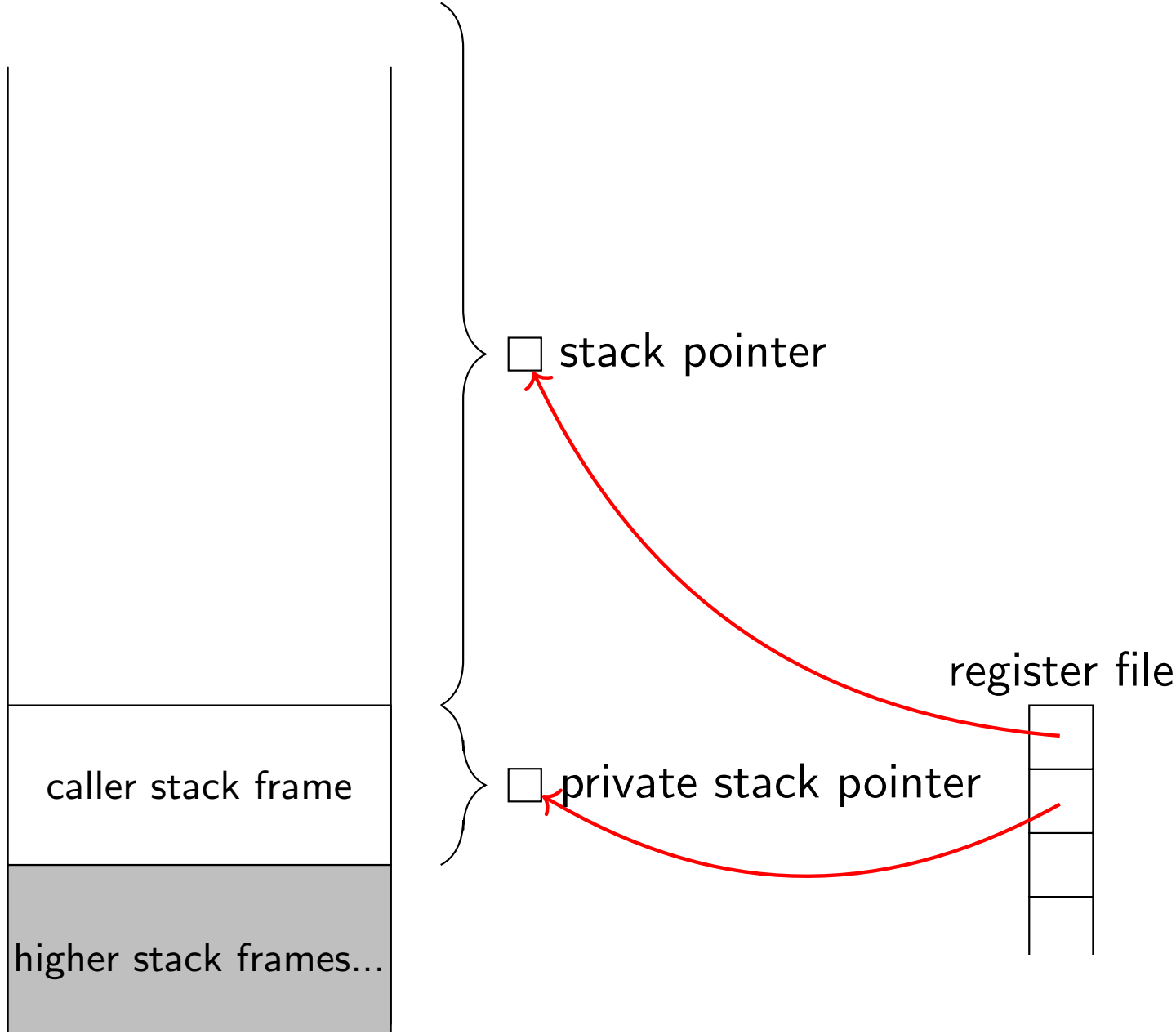
Linear Capabilities Prevent Attack 1



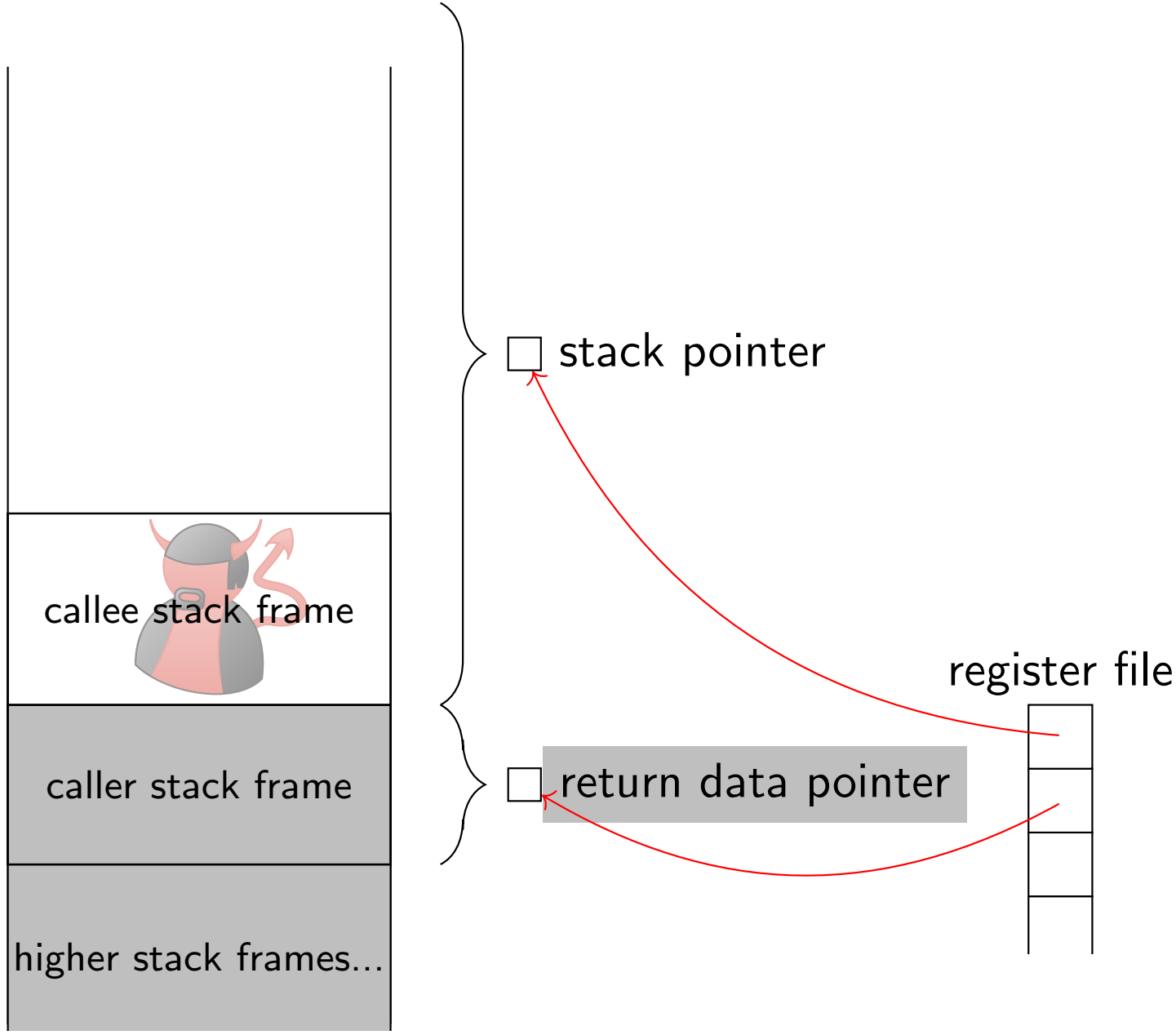
Linear Capabilities Prevent Attack 2



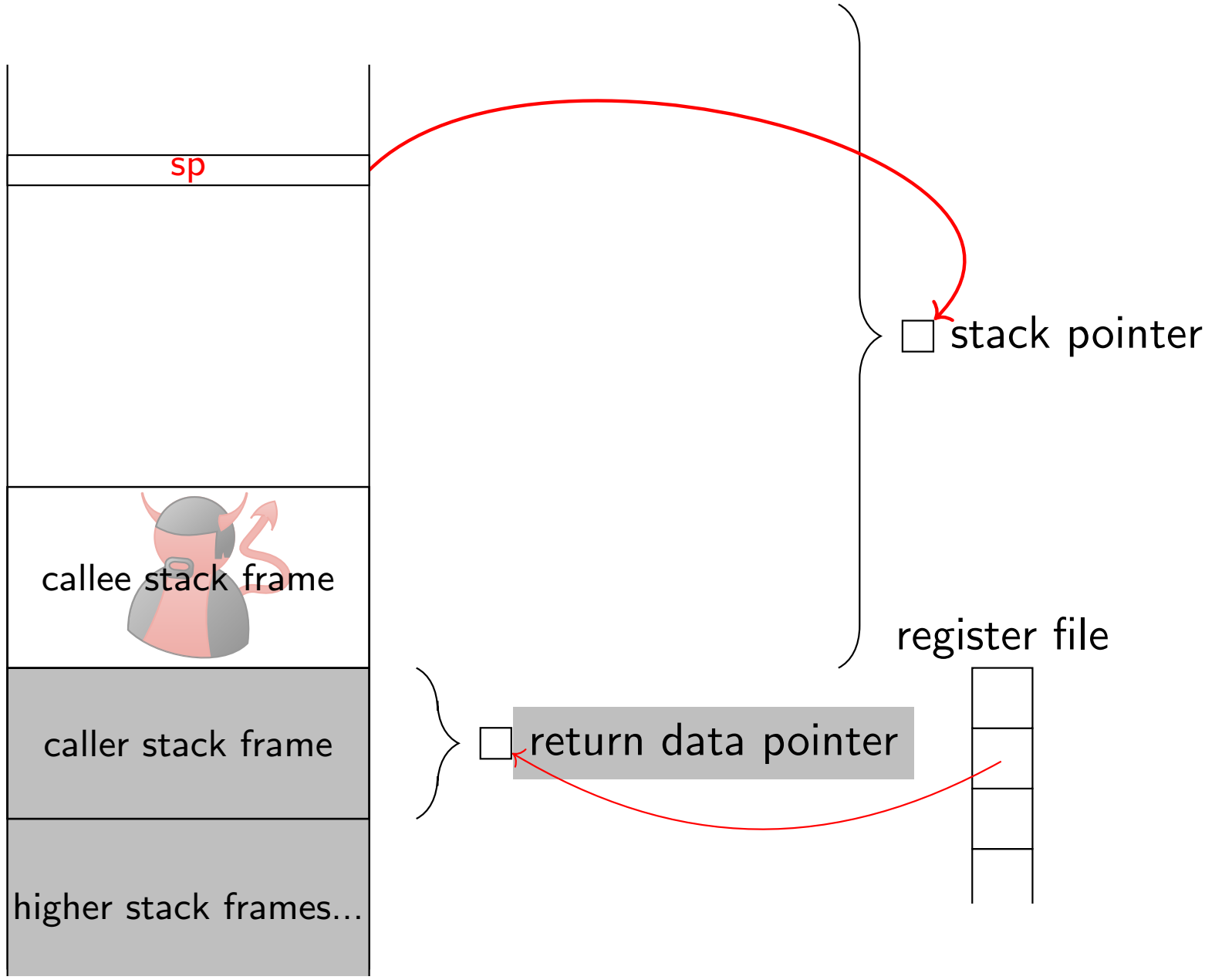
Linear Capabilities Prevent Attack 2



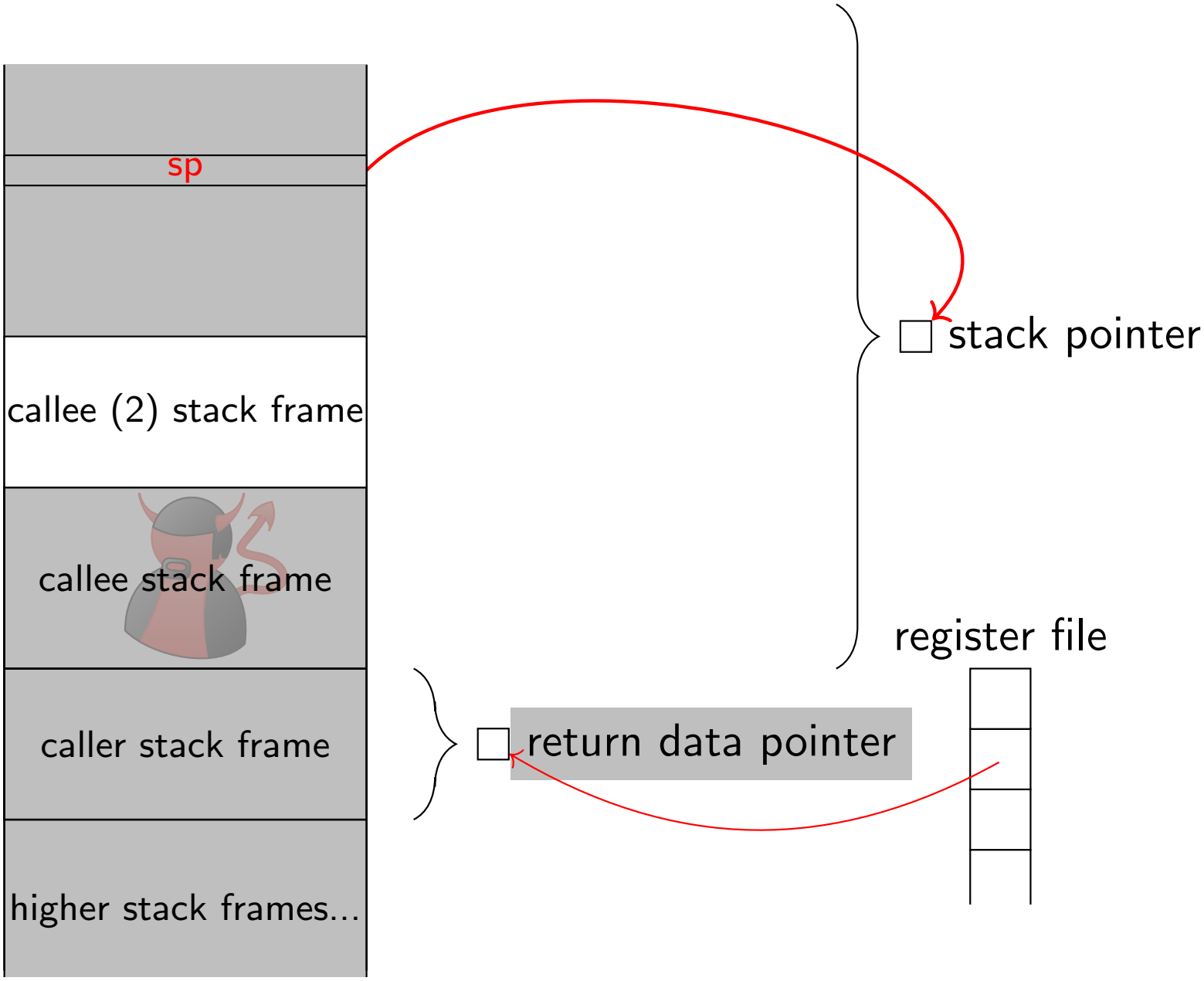
Linear Capabilities Prevent Attack 2



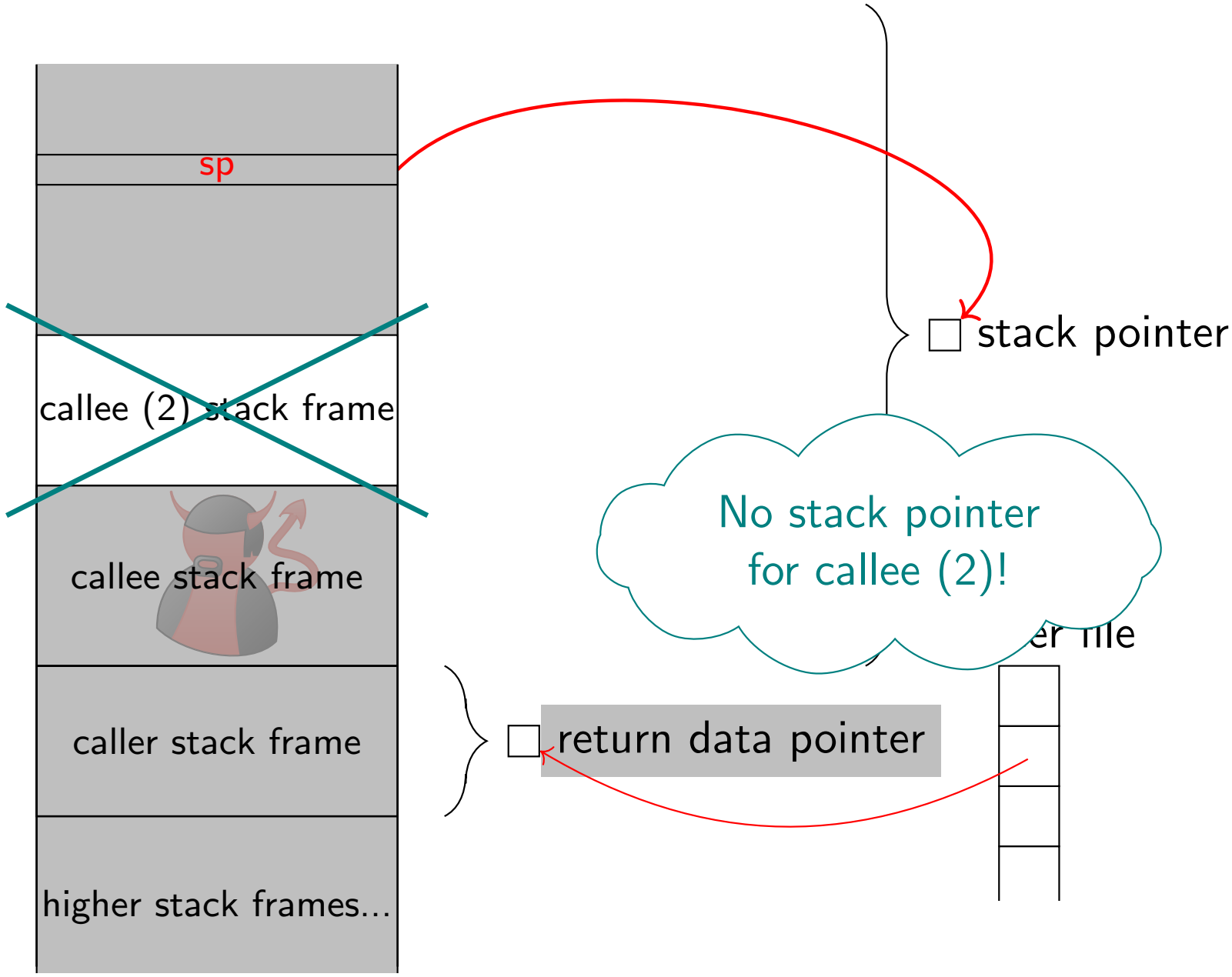
Linear Capabilities Prevent Attack 2



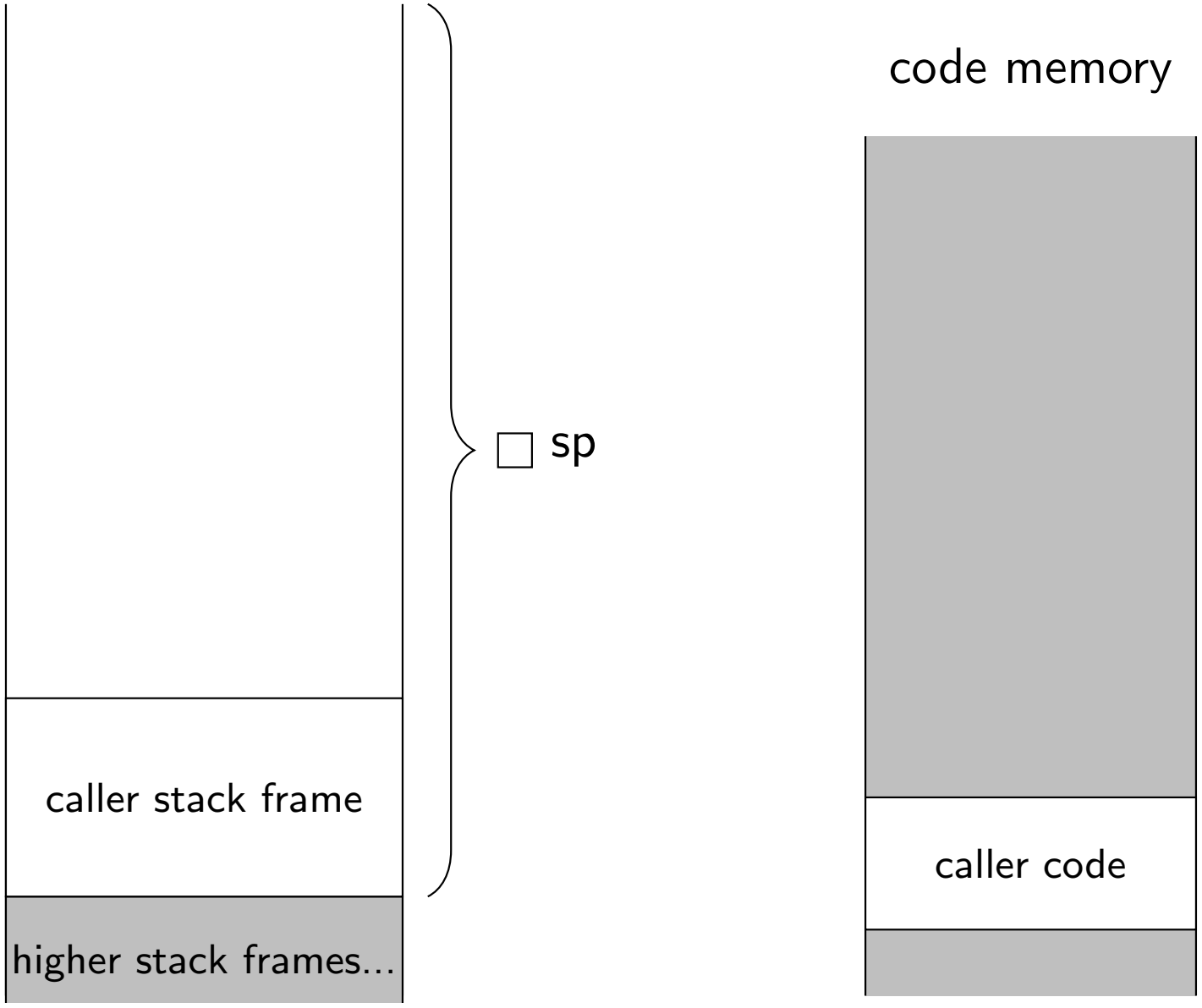
Linear Capabilities Prevent Attack 2



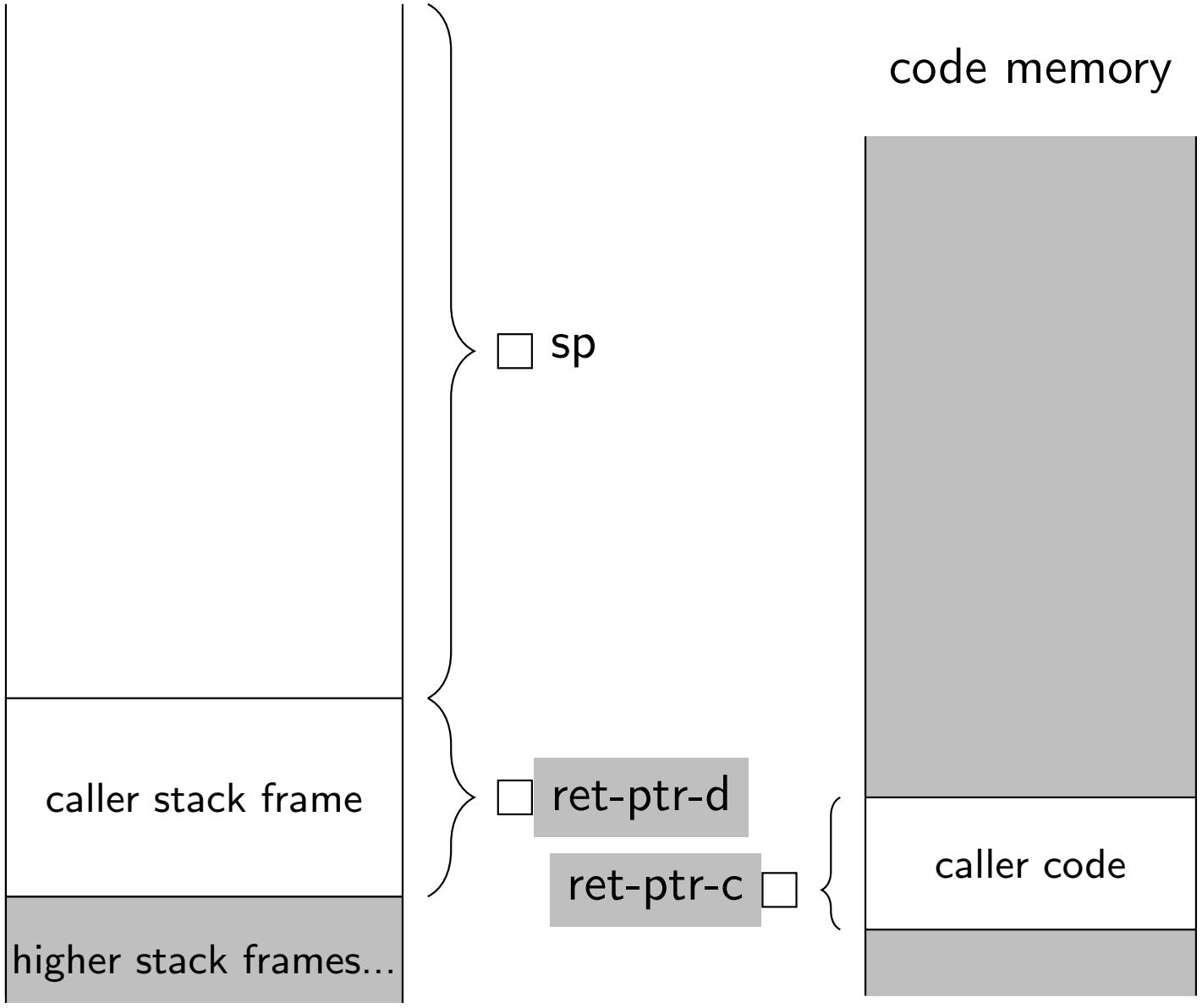
Linear Capabilities Prevent Attack 2



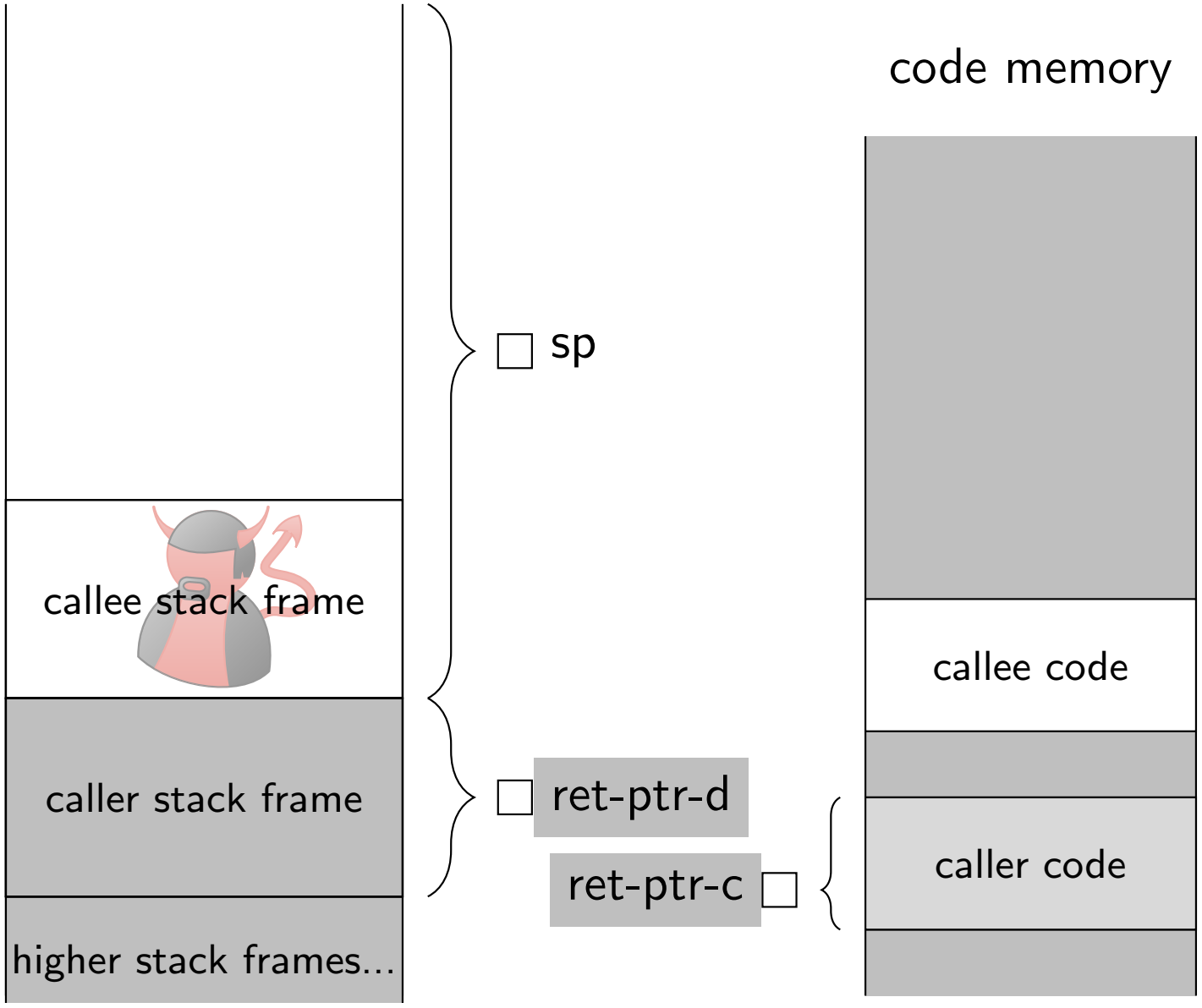
Linear Capabilities Prevent Attack 3



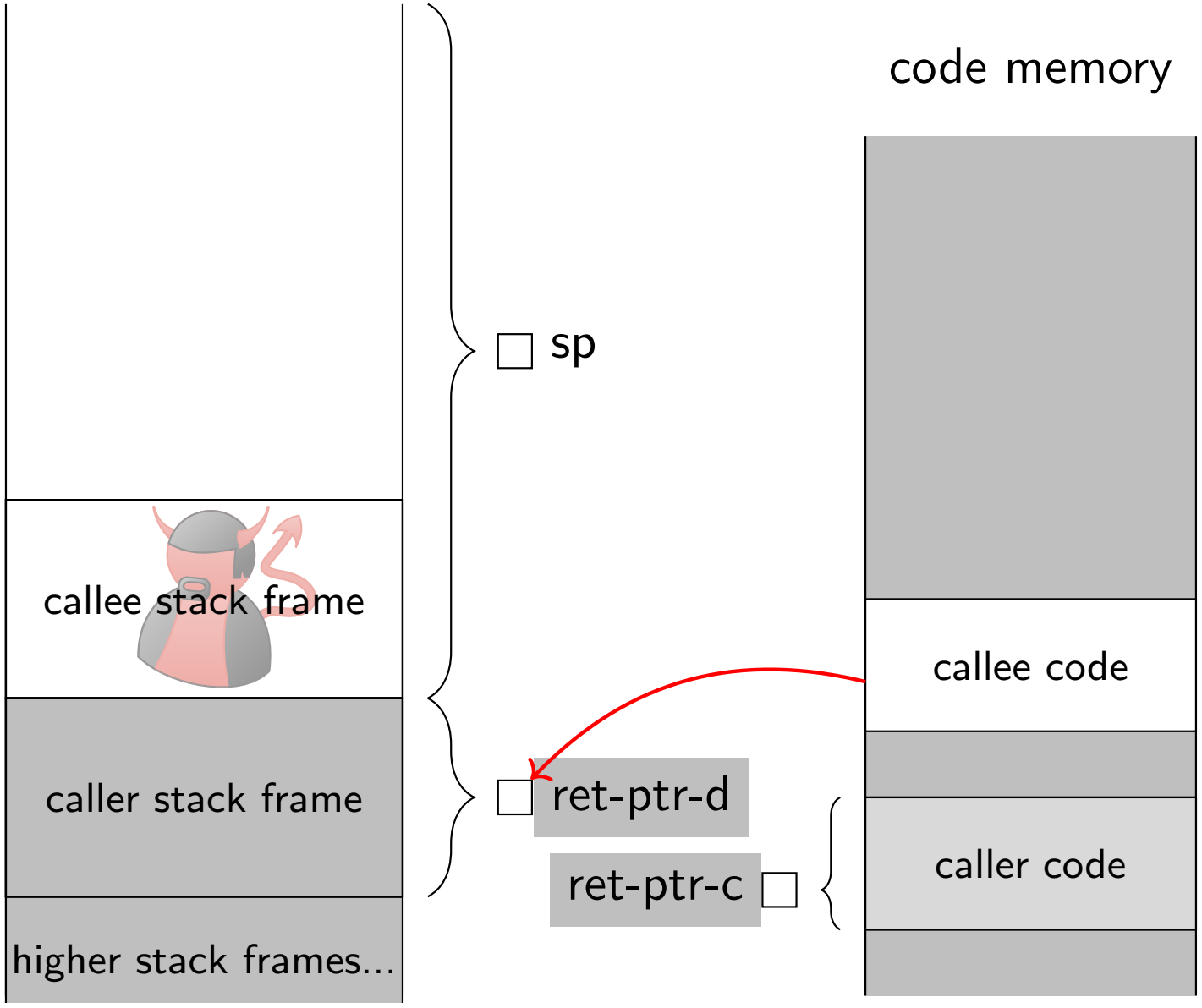
Linear Capabilities Prevent Attack 3



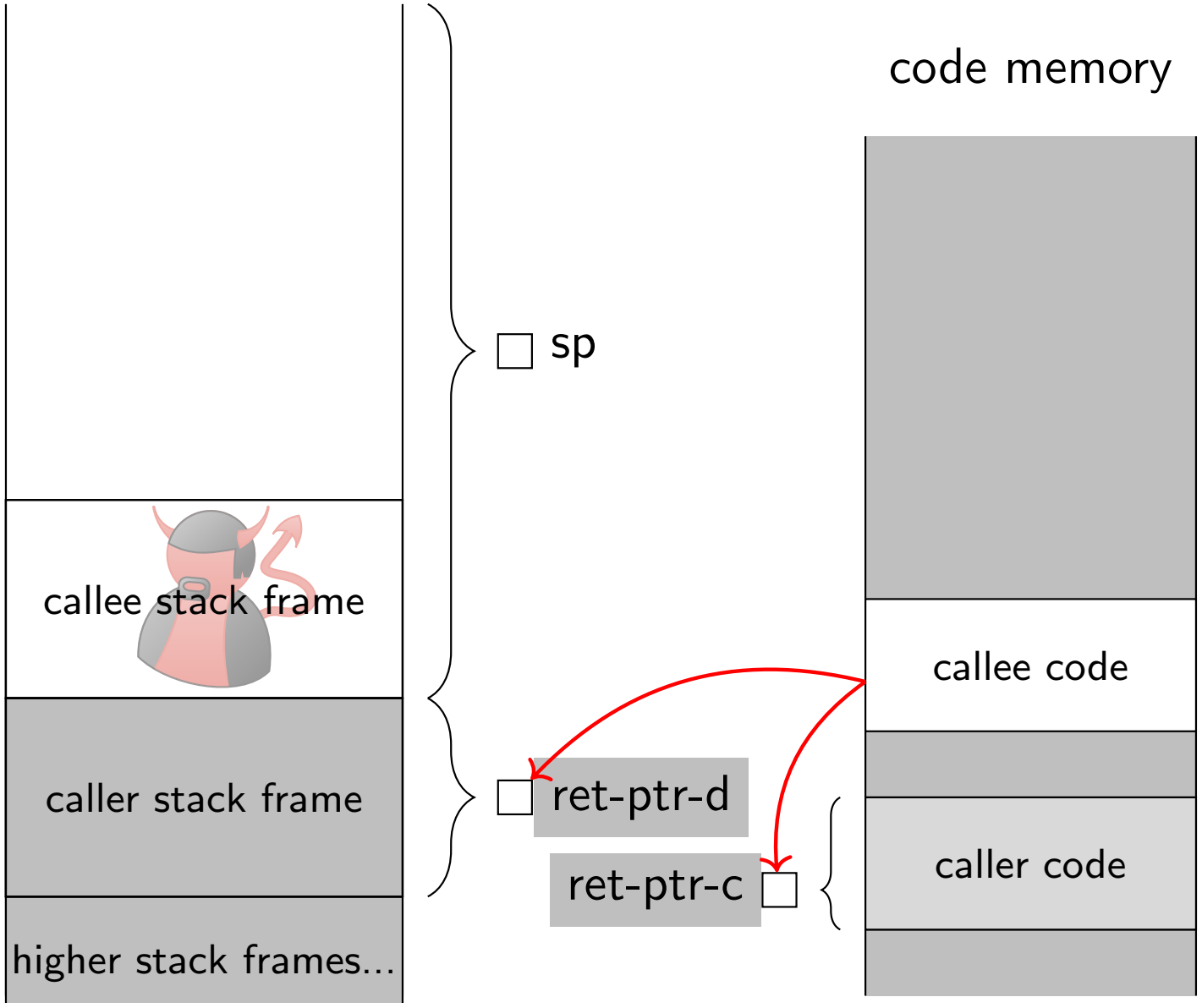
Linear Capabilities Prevent Attack 3



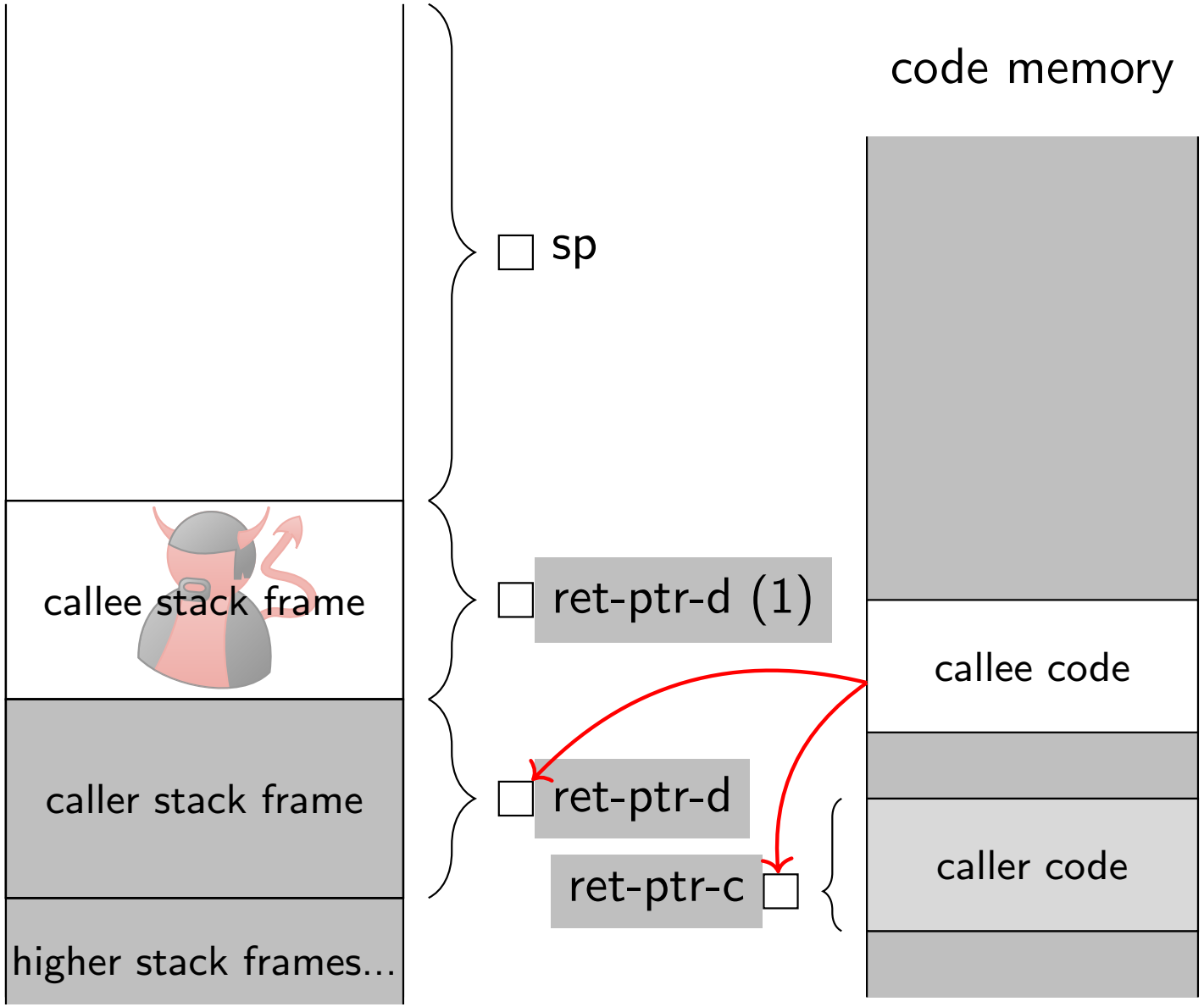
Linear Capabilities Prevent Attack 3



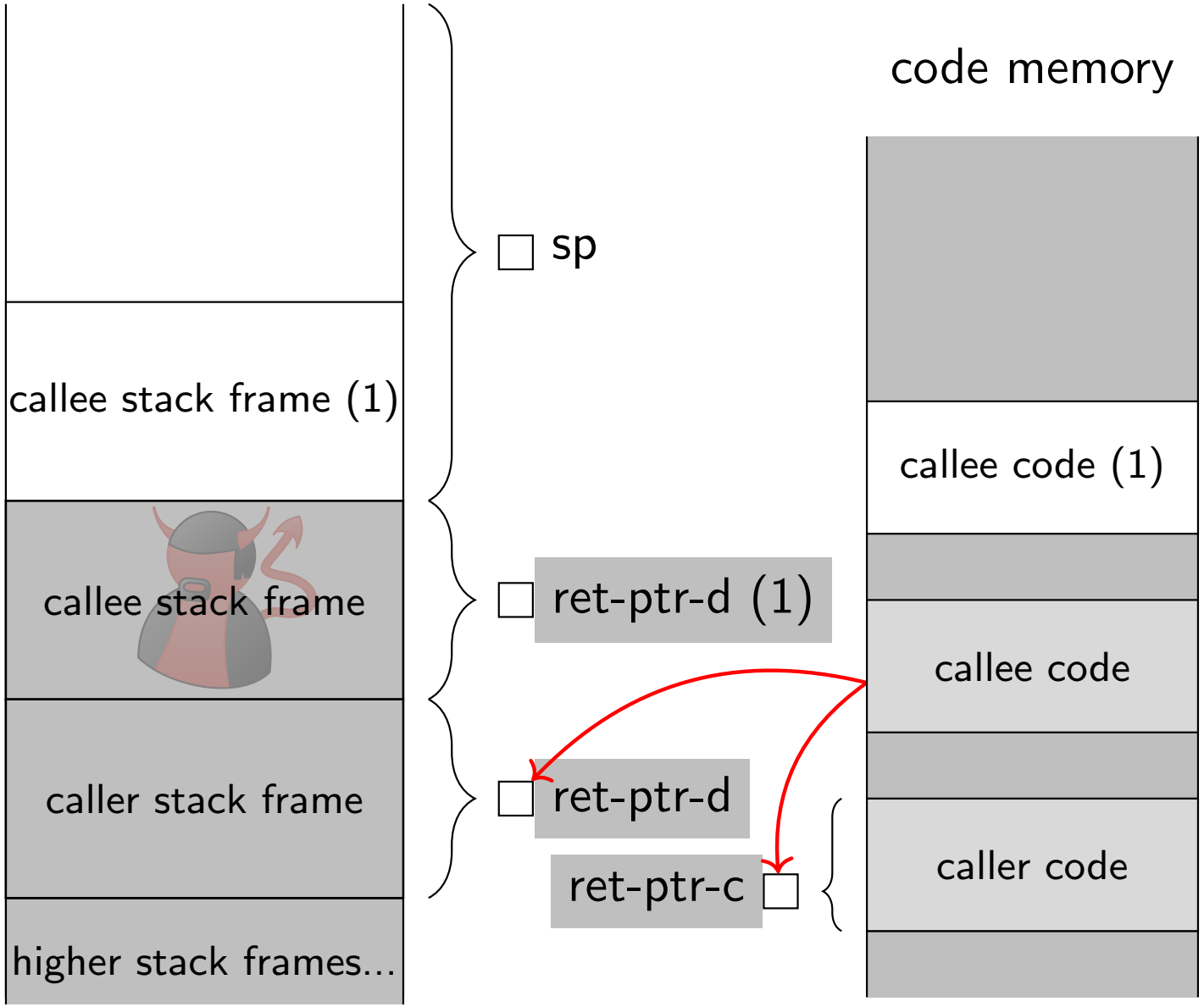
Linear Capabilities Prevent Attack 3



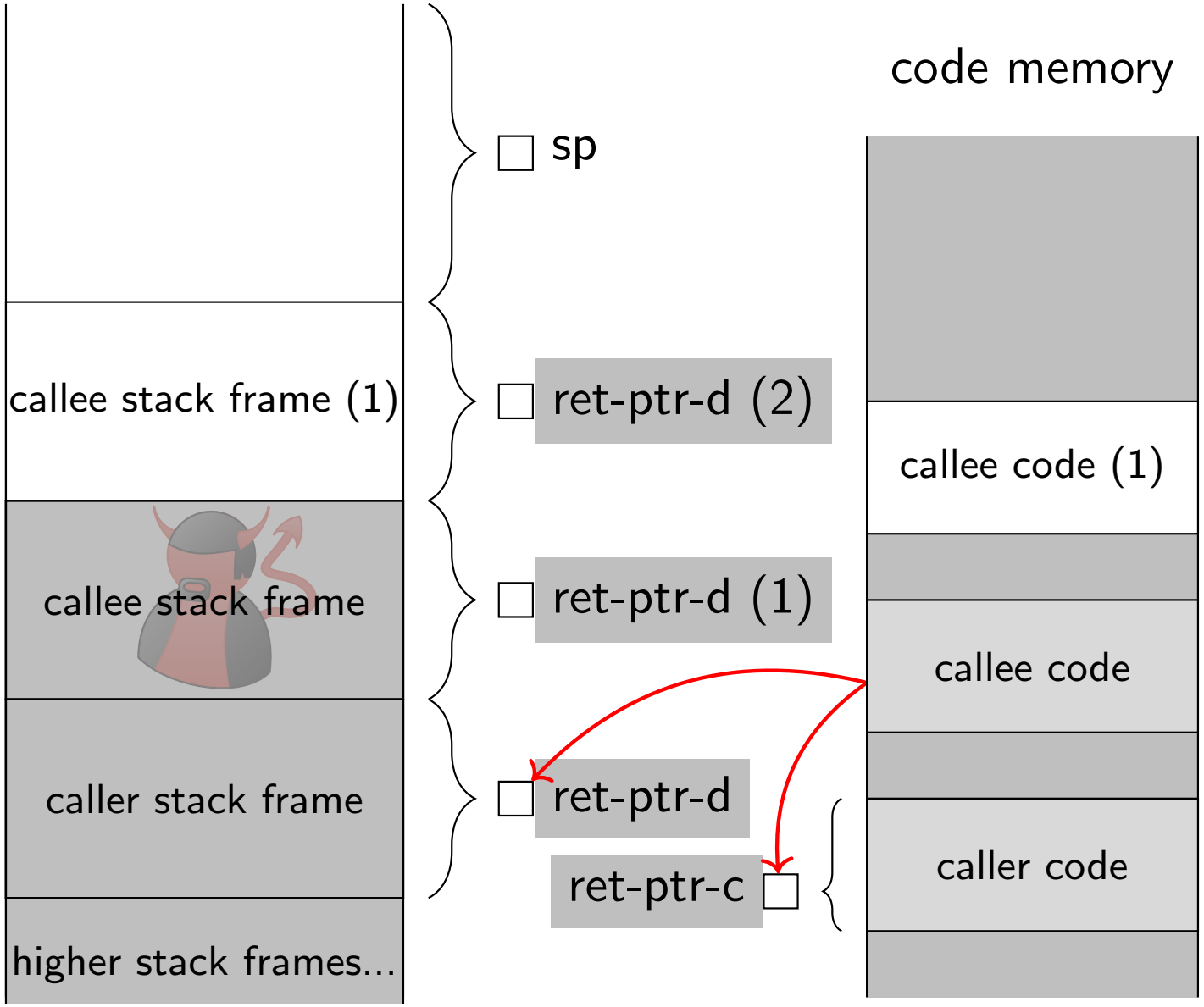
Linear Capabilities Prevent Attack 3



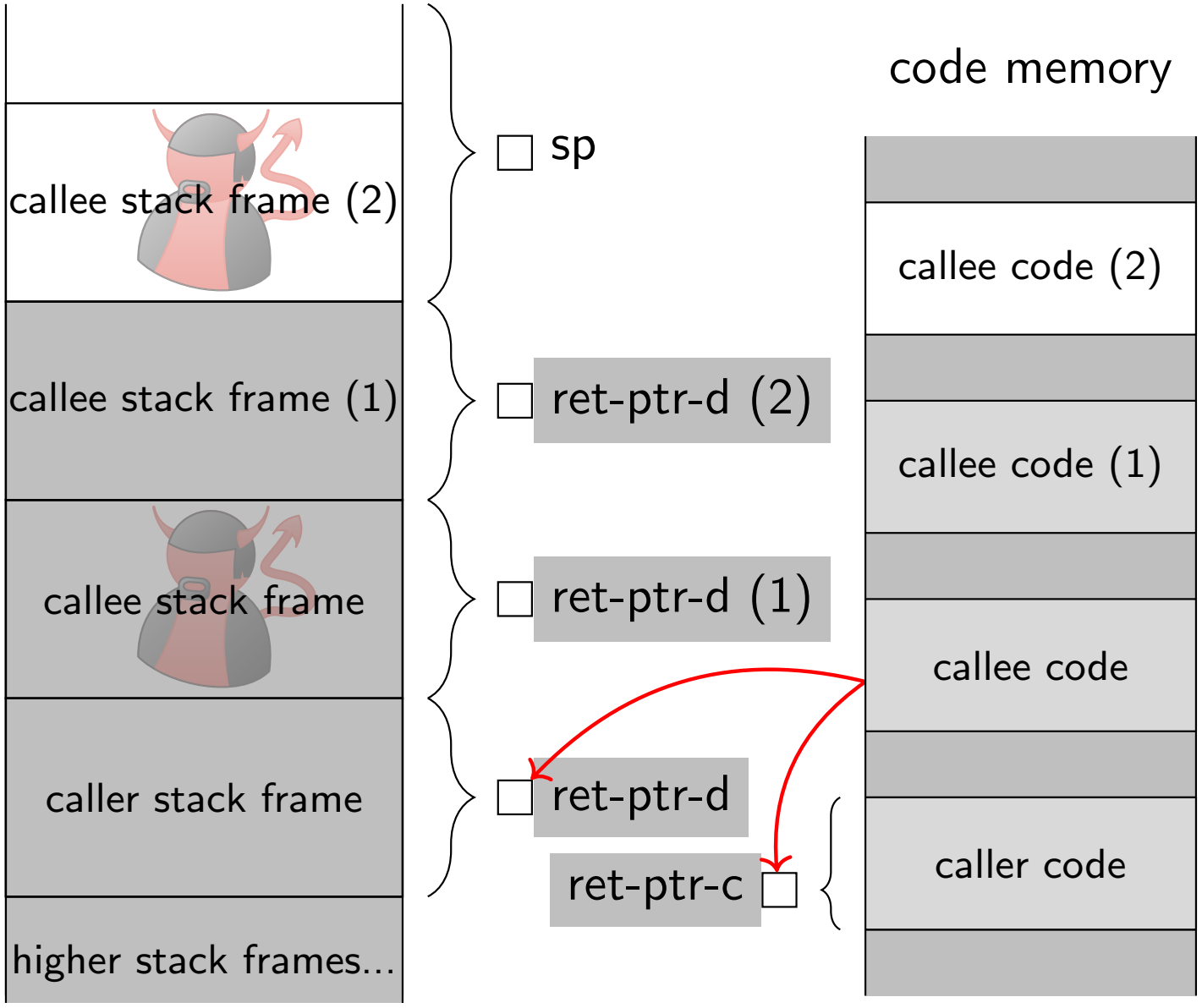
Linear Capabilities Prevent Attack 3



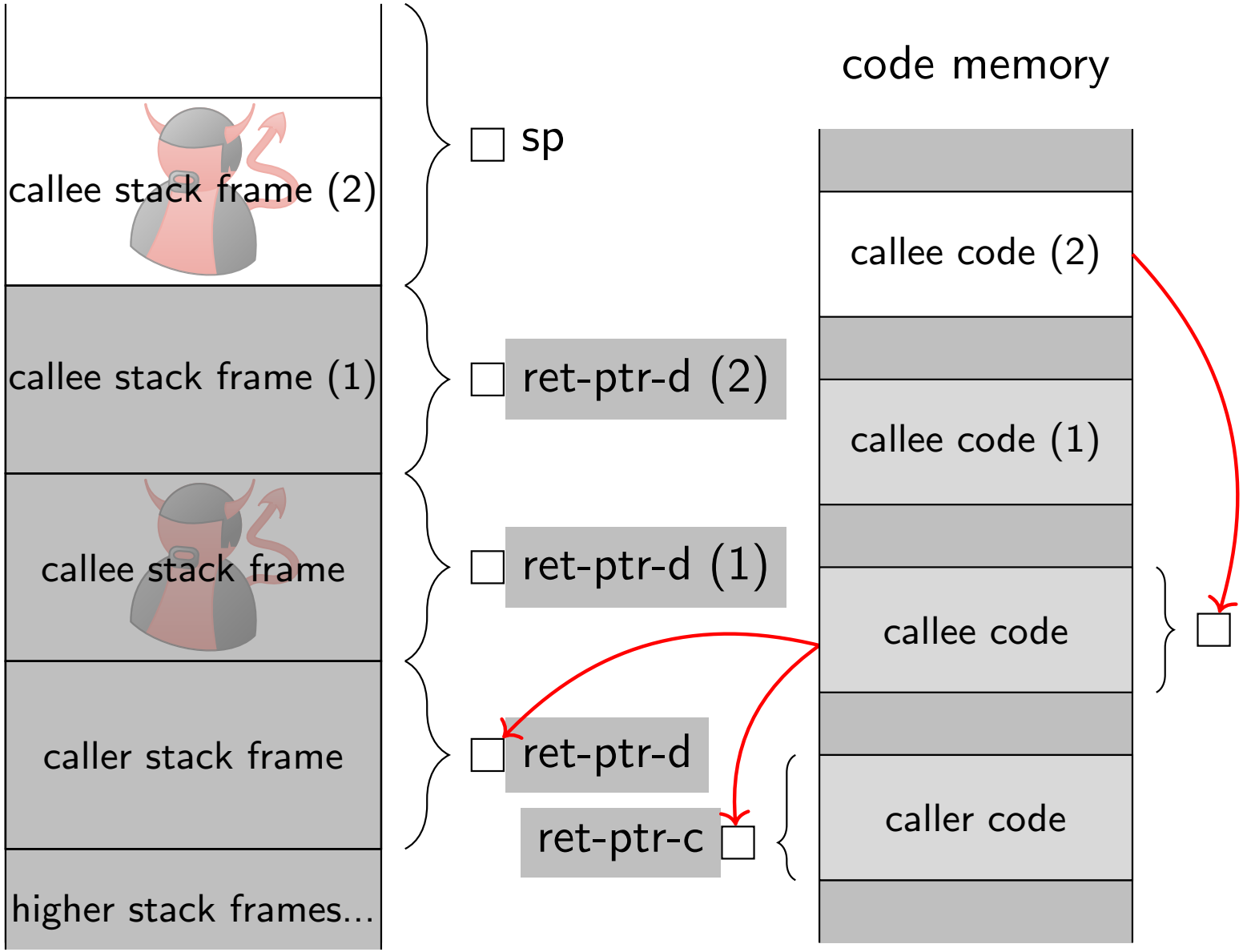
Linear Capabilities Prevent Attack 3



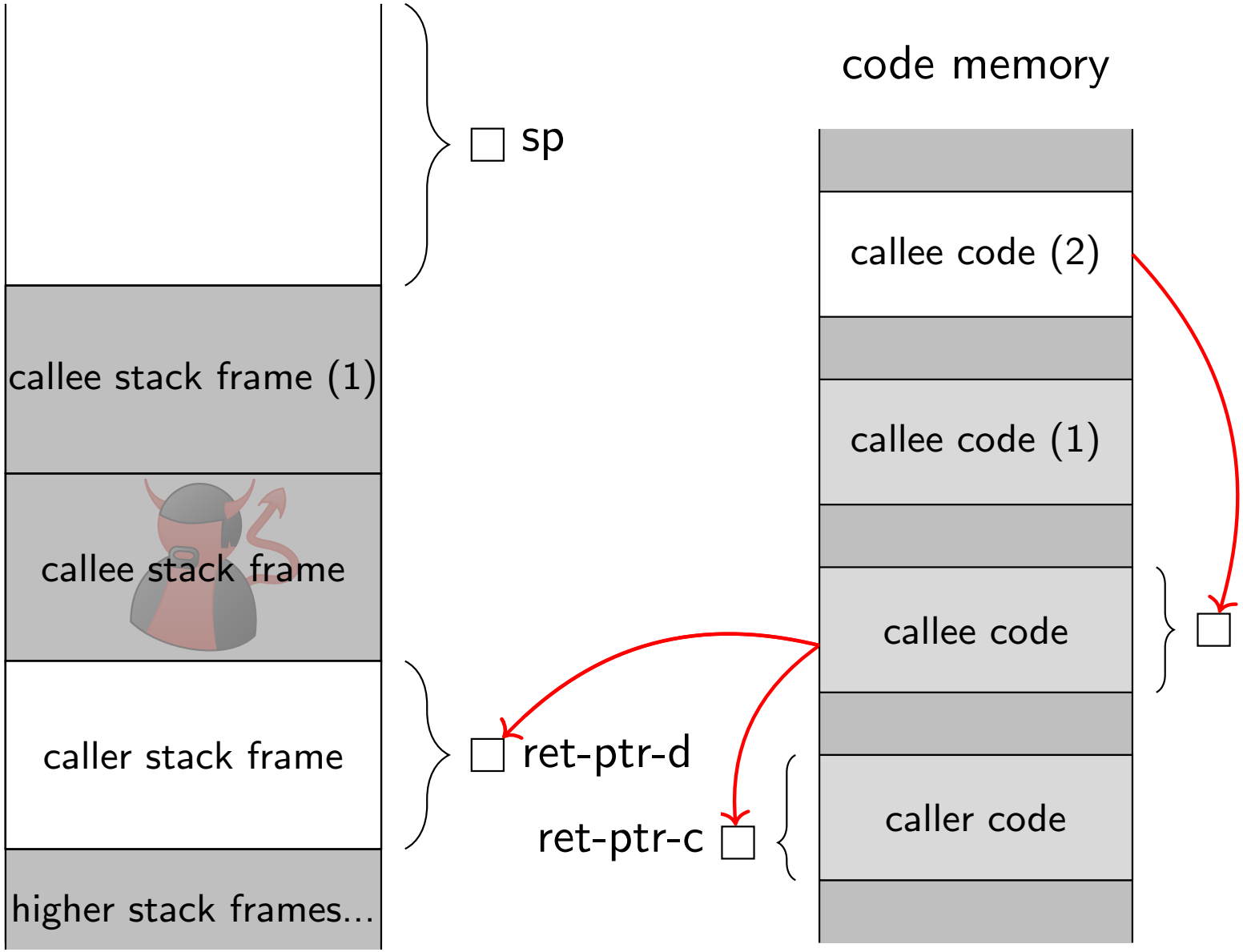
Linear Capabilities Prevent Attack 3



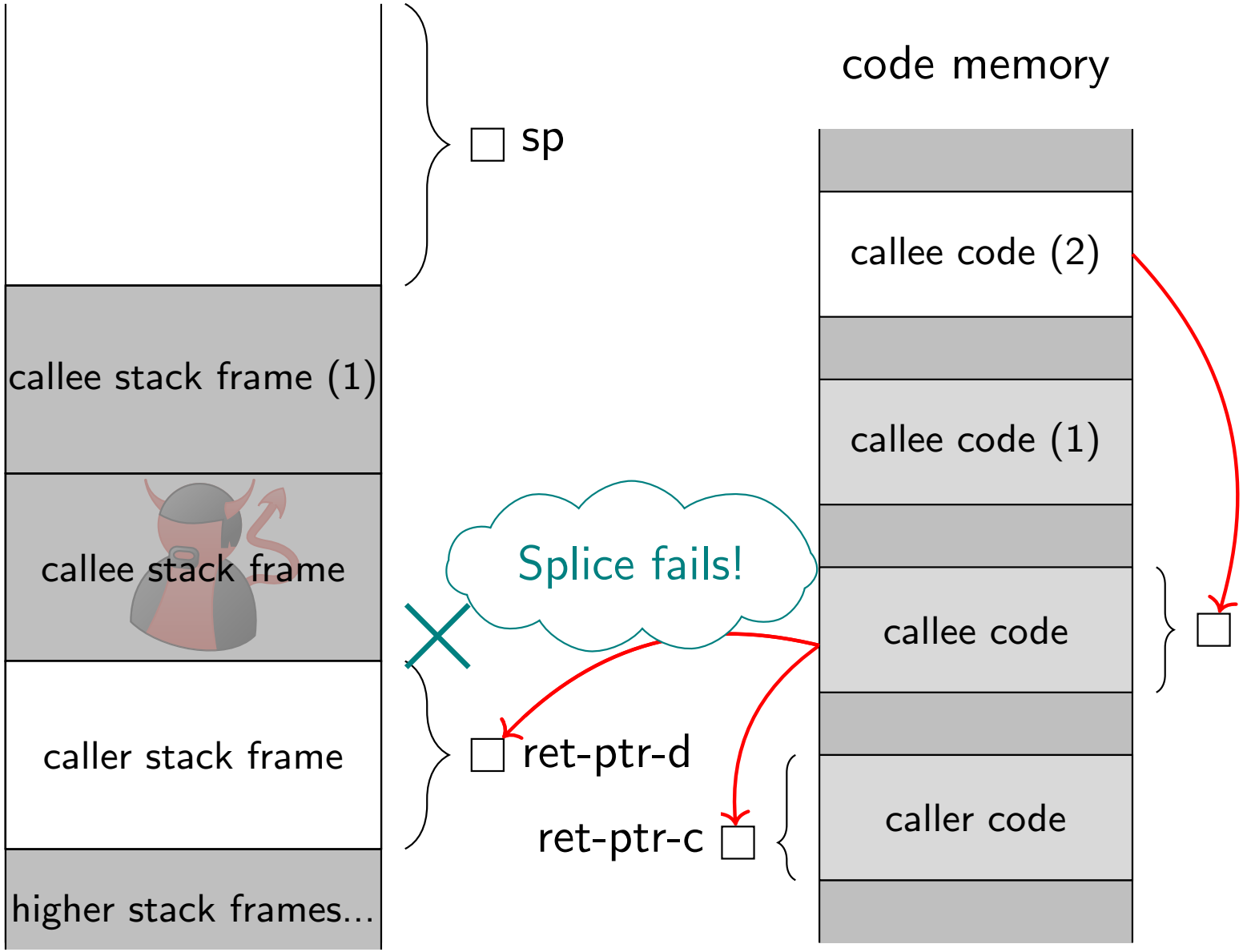
Linear Capabilities Prevent Attack 3



Linear Capabilities Prevent Attack 3



Linear Capabilities Prevent Attack 3



Stack and return pointer security

- Desirable properties:
  - Well-bracketed control flow
  - Stack frame encapsulation
- How to enforce?
  - Capabilities?
    - Not enough: three attacks!
  - CHERI Local capabilities?
    - Cannot leave registers
    - Works, but... [Skorstengaard et al., ESOP 2018]
  - CHERI Linear capabilities?
    - Non-copyable capabilities
    - Works perfectly?

Formally?

- Prove correctness?

Formally?

- Formulate correctness?
- Prove correctness?

Formally?

- Formulate correctness?
  - Fully abstract compilation
- Prove correctness?

Formally?

- Formulate correctness?
  - Idea: well-bracketed version of target assembly
  - Compiler replaces call/returns with calling convention
  - Fully abstract compilation
- Prove correctness?

Formally?

- Formulate correctness?
  - Idea: well-bracketed version of target assembly
  - Compiler replaces call/returns with calling convention
  - Fully abstract compilation
- Prove correctness?
  - Back-translation as embedding
  - Cross-language step-indexed Kripke logical relation

Conclusion

- Enforcing well-bracketedness and stack frame encapsulation on a capability machine
- Local capabilities work [Skorstengaard et al, ESOP 2018]
- Linear capabilities work better!
  - but don't exist... yet?
- Correctness as fully abstract compilation from well-bracketed version of target language.

Thanks

Questions?

Backup slide: The Awkward Example

$$\tau \stackrel{\text{def}}{=} (\mathit{unit} \to \mathit{unit}) \to \mathit{int}$$

$$e_1 \stackrel{\text{def}}{=} \begin{cases} \text{let } x \;=\; \text{ref } 0 \text{ in} \\ \lambda f.\, x := 0;\;\; f \;\text{unit}; x := 1; f \;\text{unit}; !x \end{cases}$$

$$e_2 \stackrel{\text{def}}{=} \lambda f.\, (f \;\text{unit}; f \;\text{unit}; 1)$$